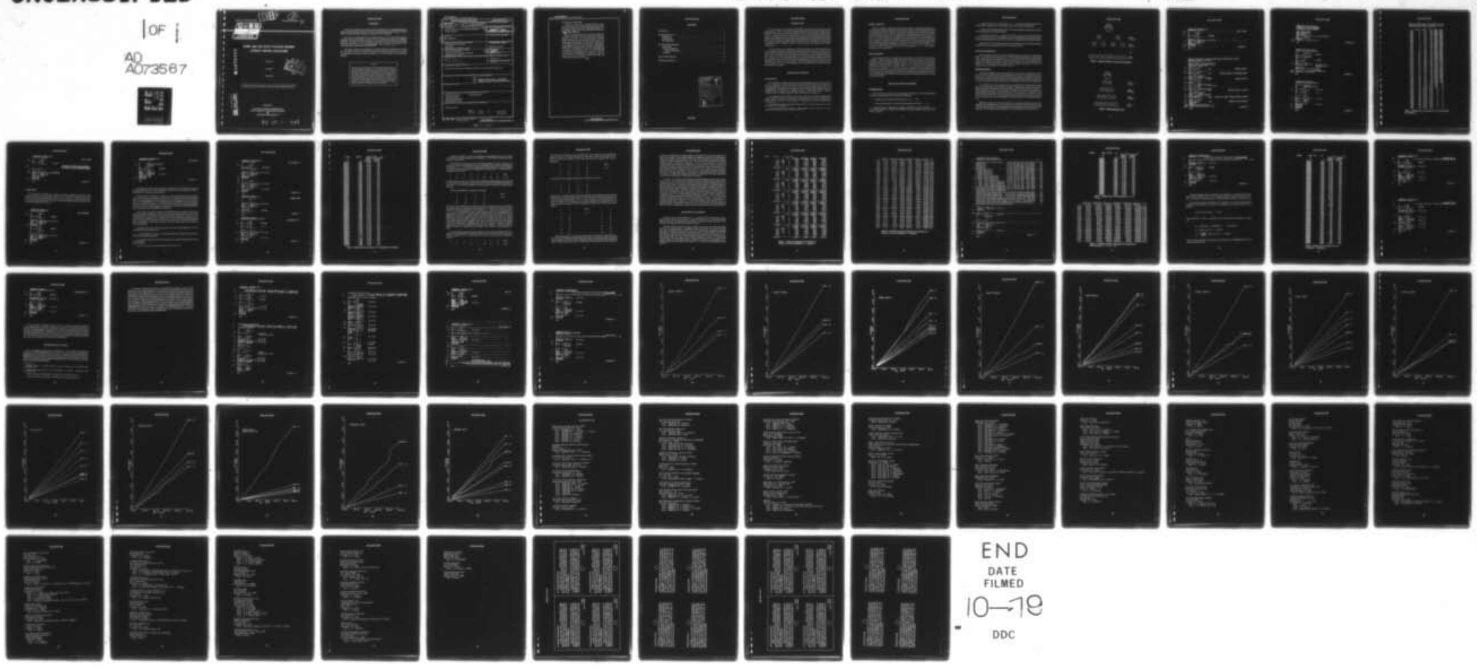
AD-A073 567 AERONAUTICAL SYSTEMS DIV WRIGHT-PATTERSON AFB OH F/G 9/2
PRIME AND PDQ SORTS - EFFICIENT MINIMAL STORAGE SORTING ALGORIT--ETC(U)
AUG 79 R R HILBRAND

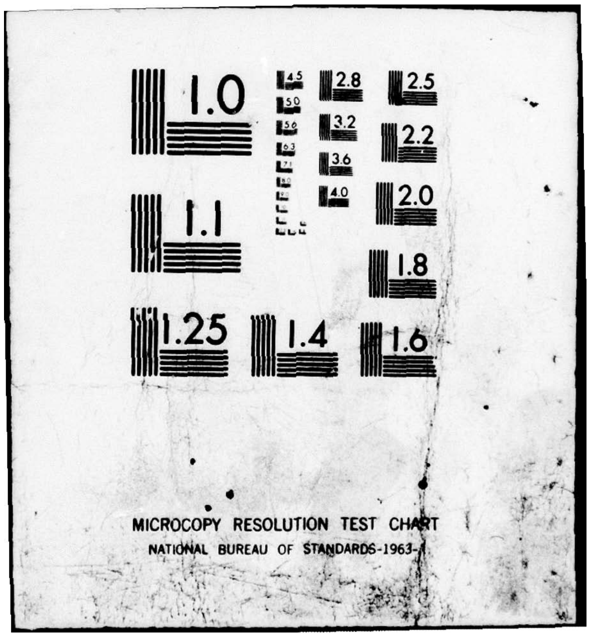
UNCLASSIFIED JTCG/AS-78-V-004 NL

1 OF 1

AD A073567



END
DATE
FILMED
10-79
DDC


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

REPORT JTCG/AS-78 V-004

FIELD OF INTEREST: 17.01
17.03

JTCG/AS

PRIME AND PDQ SORTS EFFICIENT MINIMAL STORAGE SORTING ALGORITHMS

Final Report

R. Hilbrand

August 1979

Approved for public release; distribution unlimited; statement applied August 1979.

Prepared for

THE JOINT LOGISTICS COMMANDERS
JOINT TECHNICAL COORDINATING GROUP
ON
AIRCRAFT SURVIVABILITY

FOREWORD

This report summarizes the results of research performed by the Aeronautical Systems Division, Wright-Patterson AFB, OH. The work was performed between November 1976 and March 1978, and the Project Engineer for this effort was G. B. Bennett.

The work was sponsored by the JTCG/AS as part of the 3-year TEAS (Test and Evaluation Aircraft Survivability) program. The TEAS program was funded by DDR&E/ODDT&E. The effort was conducted under the direction of the JTCG/AS Vulnerability Assessment Subgroup, as part of JTCG/AS Work Unit VA-6-02F, *Development of Aircraft Preliminary Design Assessment Methodology.*

This report presents and summarizes two sorting algorithms, PRIME Sort and PDQ Sort, developed in the Aeronautical Systems Division Computer Science Center in support of vulnerability assessment computer programs in use by the Deputy for Development Planning.

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. ~~REPORT NUMBER~~ (18) JTCG/AS-78-V-004 (19) 78-V-004 ~~2. GOVT ACCESSION NO.~~	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PRIME & PDQ Sorts - Efficient Minimal Storage Sorting Algorithms	~~5. TYPE OF REPORT & PERIOD COVERED~~ (9) Final rept. Nov[redacted]76-Mar[redacted]78
	~~6. PERFORMING ORG. REPORT NUMBER~~
7. AUTHOR(s) (10) Roy R. Hilbrand	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Aeronautical Systems Division ASD/ADDP Wright-Patterson AFB, Ohio 45433 008 800	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Work Unit VA-6-02F
11. CONTROLLING OFFICE NAME AND ADDRESS JTCG/AS Central Office Naval Air Systems Command, AIR-5204J Washington, D. C. 20361	~~12. REPORT DATE~~ (11) Aug[redacted]79
	13. NUMBER OF PAGES 54
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 63p.	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited. Statement applied August 1979.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(6) PRIME and PDQ Sorts - Efficient Minimal Storage Sorting Algorithms.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Minimal Storage Sorts	Sorting algorithms
Computer programs	
Vulnerability Assessment	
Missile endgame models	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(See reverse side)

79 09 5 027

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

008 800

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

CONTENTS

INTRODUCTION

One of the steps in most automated vulnerability assessments involves the rapid sorting of large strings of data. In the ASD (Aeronautical Systems Division) missile endgame model SESTEM,[1] for example, the aircraft components struck by the expanding fragment spray-band are calculated and stored, and then must be sorted. The sorted data are then used to compute the components struck in order of time intercepted and the resulting aircraft probability of kill. Similar data string storage and sort problems are involved in other vulnerability assessment computer programs such as FASTGEN II.[2] In SESTEM, about 40% of the computation time is typically involved with the sorting and fragment vulnerability computations.

For these reasons, the development of efficient algorithms to sort these strings of data are of considerable importance. Since large portions of the total computer run times are typically involved in performing this sorting, any improvements in them will show direct payoffs in terms of decreased run times, more rapid turnaround, greater program efficiencies, and decreased costs. The sorting algorithms in use in the vulnerability assessment programs were evaluated, and two new and more efficient programs were written. When the sorting algorithm was inserted in a new version of the SESTEM program, the average time for a run was decreased by 20 to 25%. These programs, and comparisons with some existing programs are presented in this report.

PROGRAM DEVELOPMENT

BACKGROUND

The availability of efficient general purpose sort algorithms and the sharp reduction of cost per computation of present generation computers has reduced interest in research into sort algorithms from earlier levels. Aside from the intrinsic challenge presented by the sorting problem, optimization of existing and developing applications programs written in FORTRAN IV, or similar level languages, indicate a still existing need for compact, efficient, in-line sort algorithms that are readily adaptable to specialized ends.

One approach to meeting this need is to devise a *partial sort*, an efficient algorithm by which a string of numbers is nearly sorted; and complete the sort by a method such as binary insertion, which can take advantage of a high degree of order in a number string.

[1] Aeronautical Systems Division. "SESTEM Missile Endgame Model", by G.B. Bennett, Wright-Patterson AFB, OH, ASD. September 1977. (ASD/XRH memo.)

[2] Aeronautical Systems Division. *FASTGEN II Target Description Computer Program*, by D. Cudney, Wright-Patterson AFB, OH, ASD. March 1978. (Report ASD-TR-77-24.)

PARTIAL SORTING

The basic strategy in producing partially sorted strings is to successively partition a set of numbers by exchange comparisons. Let NUM be a string of numbers of length NO equal to an integral power of two. The set of NO numbers is divided into two subsets of equal size, and the elements of one subset are compared (and exchanged when necessary) to the elements of the other subset in such a way that every element is involved in a comparison once only. As a worst case, after any comparison of sets as described, 25% of the elements less than or equal to the median value will be located below the median. Successive set divisions and comparisons continue the process, removing extreme values from the middle of the array and enabling further distribution to take place, until there are NO subsets of 1 element each. If the NO elements are distinct and represented by 1, 2, . . ., NO, Figure 1 represents the process described.

PDQ ALGORITHM

The successive division of subsets into equal subsets, as illustrated, will cause the 1 element to migrate to the proper postion in the array; but in the worst case, the 2 element will migrate toward the middle position of the array. The 2 element can be forced left by dividing the subsets unequally, as shown in Figure 2. Let the kth interval of comparison be defined as the integer part of $I_k = NO*F^k$, where $0. < F < 1$. A code to achieve the partitioning is given in Listing 1. F = .8 seems to produce a complete sort on a uniform random distribution; however, the author has not been able to characterize the optimal value for F for a particular given distribution. Observation of empirical test results suggests that certain structured distributions are difficult to sort; therefore, the PDQ algorithm may be useful as a test of randomness.

SORT CODE TESTING ALGORITHM

DISTRIBUTIONS

The sort code was tested by a control program (Listings 2A to 2D) that generates strings of a specified length from six different distributions:

1. Sorted arrays: the elements of the sequence 1, 2, 3, . . ., NO.

2. Arrays in reverse order: the elements of the sequence NO, NO-1, . . ., 2, 1.

3. Random arrays: numbers from a uniform distribution over the interval (0. , 1.). The random numbers are multiplied by 100,000 to minimize duplicate numbers when converted to integers.

4. Arrays almost in sort: the sequence, 1, 2, . . ., NO with a specified number of elements, chosen at random, set to values taken from a uniform random distribution.

5. Arrays of equal length sorted blocks: this is the distribution described in (3) sorted in successive segments of a specified length.

6. Constant value arrays: a sequence of numbers of equal value. These distributions were selected with a view to test algorithm behavior on distributions that might be encountered in practical applications, such as (3) and (4), or to demonstrate unusual characteristics.

Elapsed time to sort is measured, and a count of departures from a monotonic sequence is made. If this count is greater than 0, an error message is printed.

TESTING ENVIRONMENT

All test runs were conducted on a CDC CYBER 74, with the same level of optimization (OPT = 2). Observations of repeated runs on the CYBER 74, operating in a time-sharing mode, suggest that time measurements can vary about 20%; however~~, by~~ specifying large arrays to sort (90K), the job will be made to run on the machine in a more dedicated configuration. In this dedicated mode, elapsed times are highly reproducible. Sort times are sensitive to the values taken for F, as shown in Figure 3.

Machine Dependancy

An apparent anomally is that it takes more time to sort a sorted string than to sort a string that has an initial uniform random distribution. This seems to indicate that it takes more time to execute a branch instruction than the three arithmetic replacement statements involved in the number exchange. This is true in the aggregate for the repetitive execution of the code in the DO loop used to compare and exchange the elements of the NUM array.

The CYBER 74 is a stack machine. Up to seven words of packed instructions containing up to 28 instructions can be retained in registers constituting an instruction stack. This device can increase instruction execution speed by reducing memory references; however, a forward branch in the stack "voids the stack", therefore is an expensive operation. For this reason, PDQ will sort faster if the "less than equal" test is replaced by a "less than" test in the exchange algorithm. This increased speed is demonstrated in the decreased times required to sort a constant value array, as compared to the time required to sort an array already in sort.

Markedly different results may be expected on some other computer systems. The expected time relationship may be obtained by replacing the DO loop with an IF loop, where the branch will be in a backward direction in the instruction stack (Listing 3). Unfortunately, code optimization is not as intensive now, and overall sort times are increased.

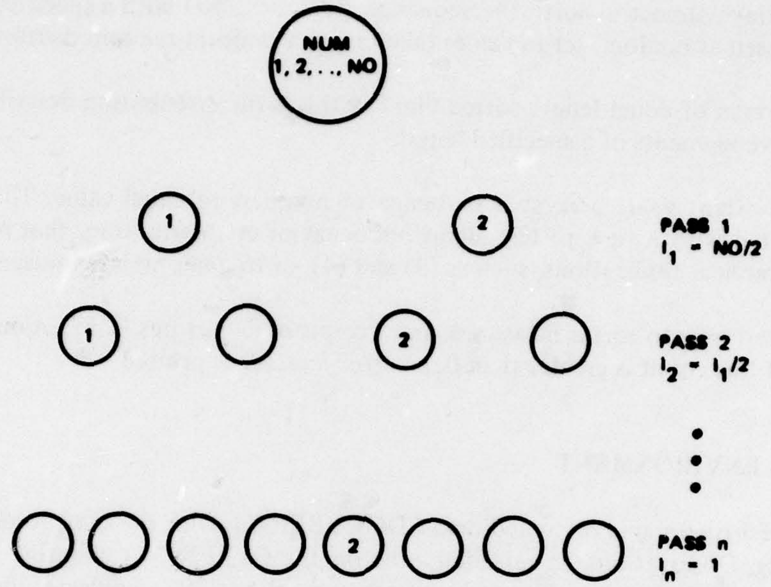


Figure 1. Successive Partition of Sets into Equal Subsets.

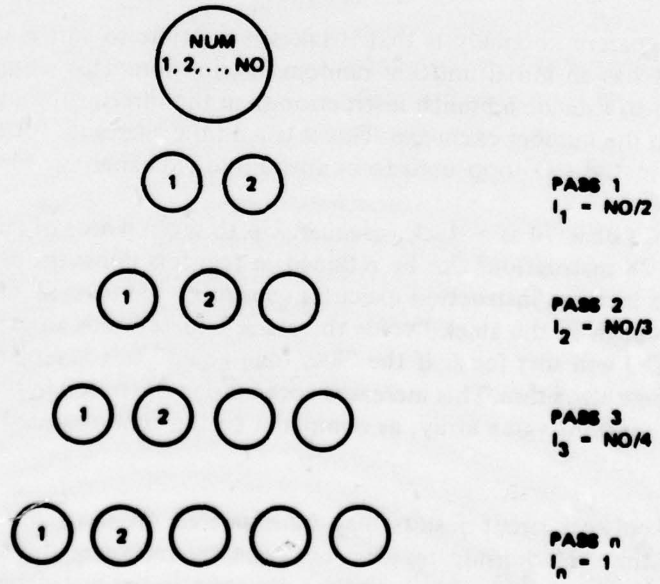


Figure 2. PDQ Partitioning Process.

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                             PDQ - BASIC
      A          = NO
 10   A          = A*.8
      I          = A
      IF(I .LE. 0)            RETURN
      K          = NO - I
      DO 15 J = 1,K
      IF(NUM(J).LE.NUM(I+J)) GO TO 15
      MAX        = NUM(  J)
      NUM(  J)=  NUM(I+J)
      NUM(I+J)=  MAX
 15   CONTINUE
      GO TO 10
C                                                               LISTING 1
      END
```

```
      PROGRAM SORT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,PUNCH)
      COMMON TIME(7),NO,A(90000)
      DATA ICEED/78/
      READ (5,100) IP1,IP2,IP3,ALF,M
      WRITE(6,115) IP1,IP2,IP3,ALF,M
      DO 80 NO= IP1,IP2,IP3
      DO 10 N = 1,NO
 10   A(N)       = N
      CALL TEST(1,              40H                          SORTED ARRAYS)
      DO 20 N = 1,NO
 20   A(N)       = NO - N + 1
      CALL TEST(2,              40H          ARRAYS SORTED IN REVERSE ORDER)
      CALL RANSET(ICEED)
      DO 30 N = 1,NO
 30   A(N)       = RANF(B)*100000.
      CALL TEST(3,              40H                          RANDOM ARRAYS)
      DO 40 N = 1,NO
 40   A(N)       = N
      DO 45 I = 1,M
      N          = RANF(B)*100000.
 42   IF(N .LT. NO)             GO TO 45
      N          = N/2
      GO TO 42
 45   A(N)       = RANF(B)*100000.
      CALL TEST(4,              40H                  ARRAYS ALMOST IN SORT)
      CALL RANSET(ICEED)
      DO 50 N = 1,NO
 50   A(N)       = RANF(B)*100000.
      CALL STRING(M)
      CALL TEST(5,              40H  ARRAYS OF EQUAL LENGTH SORTED BLOCKS)
      DO 60 N = 1,NO
 60   A(N)       = 8.
      CALL TEST(6,              40H                  CONSTANT VALUE ARRAYS)
 80   CONTINUE
      STOP
 100  FORMAT(3I10,A10,4I10 )
 115  FORMAT(1H1,3I10,2X,A10,4I10 )
C                                                              LISTING 2A
      END
      SUBROUTINE TEST(K,ALF)
      COMMON TIME(7),NO,A(90000)
      DIMENSION ALF(5),NUM(90000)
      EQUIVALENCE (A(1),NUM(1))

      CALL TYME (K,K)
      CALL SORT(NUM,NO)
      CALL ETIME(K,K)
      CALL CHECK
      WRITE(6,200) (ALF(I),I=1,4),NO,TIME(K)
      N1        = NO - 8
      CALL OUT(N1,NO)
      RETURN
  200 FORMAT(1H ,4A10,I8,F10.2 )
C                                                           LISTING 2B
      END
```

```
      SUBROUTINE TYME(N1,N2)
      COMMON TIME(7),NO,A(90000)

      TP        = SECOND(B)
                                     RETURN
      ENTRY ETIME
      TC        = SECOND(B)
      TIME(N1)= TC - TP
      TP        = TC
                                     RETURN
      ENTRY CHECK
      IE        = 0
      DO 20 J = 2,NO
   20 IF(A(J-1) .GT. A(J))           IE = IE + 1
      IF(IE .NE. 0)                  WRITE(6,100) IE
                                     RETURN
      ENTRY OUT
      WRITE(6,300) (A(N),N=N1,N2)
                                     RETURN
  100 FORMAT(1H , 35X,12HERROR COUNT=,I6)
  300 FORMAT(1H+,65X,9F7.0 )
C                                                           LISTING 2C
      END
```

```
      SUBROUTINE STRING(M)
      COMMON TIME(7),NO,A(90000)

      DO 30 N1= 1,NO,M
      N2        = MIN0(N1+M-1,NO)
      I         = N2 - N1 + 1
      B         = I
   10 IF(I .LE. 1)                   GO TO 30
      B         = B*.425
      I         = B
      IF(I .LT. 1)                   I = 1
      K         = N2 - I

      DO 25 J = N1,K
      L         = J
   15 IF(A(L) .LE. A(I+L))           GO TO 25
      BIG       = A(  L)
      A(  L)    = A(I+L)
      A(I+L)    = BIG
      L         = L - I
      IF(L .GE. N1)                  GO TO 15
   25 CONTINUE
      GO TO 10
   30 CONTINUE
      RETURN
C                                                           LISTING 2D
      END
```

```
TIME IN SECONDS AND ERROR COUNTS FOR TWO
SUCCESSIVE PDQ SORTS ON A UNIFORM RANDOM
DISTRIBUTION OF 10000 NUMBERS

    F   PASSES   TIME1   TIME2   ERR1   ERR2
 .200       5     .27     .26    5030   5009
 .210       5     .24     .22    5011   5023
 .220       6     .28     .27    4995   5014
 .230       6     .30     .28    4984   4991
 .240       6     .28     .26    5019   4389
 .250       6     .29     .27    5016   5009
 .260       6     .28     .27    5005   4981
 .270       7     .32     .32    4977   4970
 .280       7     .33     .31    5005   4956
 .290       7     .32     .32    5021   4993
 .300       7     .33     .31    5029   4921
 .310       7     .32     .31    5016   4989
 .320       8     .37     .35    4960   4945
 .330       8     .36     .37    5020   4975
 .340       8     .37     .34    4948   4995
 .350       8     .37     .35    4969   4974
 .360       9     .40     .39    4947   4932
 .370       9     .40     .39    4941   4967
 .380       9     .41     .40    4941   4898
 .390       9     .42     .40    4985   5021
 .400      10     .45     .44    4989   4897
 .410      10     .45     .44    4948   4924
 .420      10     .44     .43    4978   4958
 .430      10     .45     .42    4996   4954
 .440      11     .49     .48    4981   4806
 .450      11     .51     .48    4983   4830
 .460      11     .49     .46    4924   4838
 .470      12     .54     .52    4943   4760
 .480      12     .53     .50    4960   4793
 .490      12     .54     .51    4912   4773
 .500      13     .56     .56    4893   4777
 .510      13     .56     .54    4906   4724
 .520      14     .61     .59    4892   4438
 .530      14     .61     .60    4897   4475
 .540      14     .61     .58    4912   4572
 .550      15     .64     .63    4685   4123
 .560      15     .65     .63    4824   4320
 .570      16     .69     .67    4643   3867
 .580      16     .69     .66    4790   4336
 .590      17     .70     .69    4562   3194
 .600      18     .76     .70    4103   2451
 .610      18     .73     .75    4269   2545
 .620      19     .80     .79    4135   2556
 .630      19     .79     .78    4244   3117
 .640      20     .81     .80    3200   1212
 .650      21     .85     .86    3319   1073
 .660      22     .90     .89    1888     96
 .670      22     .88     .93    2162    293
 .680      23     .91     .95    1273      9
 .690      24     .94    1.04    1242     18
 .700      25     .99    1.13     401      0
 .710      26    1.00    1.15     354      0
 .720      28    1.06    1.30      23      0
 .730      29    1.10    1.37      20      0
 .740      30    1.15    1.48       4      0
 .750      32    1.20    1.52       0      0
 .760      33    1.24    1.61       0      0
 .770      35    1.30    1.71       0      0
 .780      37    1.44    1.80       0      0
 .790      39    1.49    1.91       0      0
 .800      41    1.55    2.00       0      0
 .810      43    1.61    2.10       0      0
 .820      46    1.76    2.21       0      0
 .830      49    1.90    2.43       0      0
 .840      52    1.96    2.51       0      0
 .850      56    2.13    2.73       0      0
 .860      61    2.34    3.01       0      0
 .870      66    2.52    3.21       0      0
 .880      72    2.77    3.53       0      0
 .890      79    3.01    3.75       0      0
 .900      87    3.28    4.22       0      0
```

Figure 3. Elapsed Time as a Function of F for the PDQ Partial Sort (Listing 1).

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                           PDQ IF LOOP
      A          = NO
 10   A          = A*.8
      I          = A
      IF(I .LE. 0)             RETURN
      K          = NO - I
C                                       LOOP WILL EXECUTE ONCE AS WITH A
C                                       FORTRAN4 DO - LOOP, REGUARDLESS
C.                                      OF INITIAL INDEX PARAMETER VALUES
      J          = 1
      GO TO 20
 15   J          = J + 1
      IF(J .GT. K)             GO TO 25
 20   IF(NUM(J).LE.NUM(I+J))   GO TO 15
      MAX        = NUM(   J)
      NUM(   J)= NUM(I+J)
      NUM(I+J)= MAX
      GO TO 15
 25   CONTINUE
      GO TO 10
C                                                           LISTING  3
      END
```

VARIATIONS

The simplicity of the PDQ code lends it to numerous variations; for instance, PDQ modules can be stacked with different values for F in each module. Changing the direction of the comparisons on alternate passes causes sorting to occur in fewer passes but at the cost of increased complexity (Listing 4). Listing 5 is a format of PDQ using a partitioning scheme involving the logarithm to the base 12.

```
      SUBROUTINE SORT(F)
      COMMON NUM(10000),NO
C                                                           PDQ ALTERNATE
      A          = NO
 10   A          = A*F
      I          = A
      IF(I .LE. 0)             RETURN
      K          = NO - I

      DO 15 J = 1,K
      IF(NUM(J).LE.NUM(I+J))   GO TO 15
      MAX        = NUM(   J)
      NUM(   J)= NUM(I+J)
      NUM(I+J)= MAX
 15   CONTINUE
      A          = A*F
      I          = A
      IF(I .LE. 0)             RETURN
      K          = NO - I
      L          = K + 1

      DO 20 J = 1,K
      L          = L - 1
      IF(NUM(L).LE.NUM(I+L))   GO TO 20
      MAX        = NUM(   L)
      NUM(   L)= NUM(I+L)
      NUM(I+L)= MAX
 20   CONTINUE
      GO TO 10
C                                                           LISTING  4
      END
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                          PDQ LOG12
      A          = NO
      P          = ALOG10(A)/1.07918
 10   P          = P - .1
      A          = 12.**P
      I          = A
      IF(I .LE. 0)               RETURN
      K          = NO - I
      DO 15 J    = 1,K
      NUM1       = NUM(  J)
      NUM2       = NUM(I+J)
      IF(NUM1 .LT. NUM2)         GO TO 15
      NUM(  J)= NUM2
      NUM(I+J)= NUM1
 15   CONTINUE
      GO TO 10
C                                                          LISTING 5
      END
```

The original intention of the sort strategy is achieved, with the assurance of a sort by following PDQ with a sort by direct insertion. The insertion code is derived from PDQ by the addition of a few lines of code (Listing 6).

Insertion means the addition of numbers to an existing sorted string (which initially may be of length 1) by inserting the new element into, or at the ends of the sorted string to form a new sorted string of increased length. For any distribution, the sort is completed in one pass. An INSERTION sort is to be distinguished from a BUBBLE sort where successive elements are selected and added to one end of a string, initially of length 0. The BUBBLE sort may require up to NO-1 passes to complete the sort. Tests show the INSERTION sort to be more efficient than the BUBBLE sort (Listing 7).

If the features of the PDQ sort and the INSERTION sort are combined, a particularly efficient algorithm resembling the SHELL sort is obtained (Listing 8). The specification of an optimal sequence of intervals to control partitioning is a difficult task; however, if a geometric sequence is assumed, an F can empirically be found which will yield minimum sort times for a given distribution (Figure 4). If two or more values of F produce minimal elapsed times, the smallest of these values should be selected to reduce sort times on sorted or nearly sorted strings.

The SHELL sort seems to have been intended as a type of merge algorithm.[3] The term "merge" may be used in several ways:

1. The combination of two or more sorted strings into a resultant sorted string irrespective of any particular algorithm.

2. A specific method by which sorted strings can be combined efficiently into resultant sorted strings.

[3]Gotlieb, C.C. Sorting on Computers Communications, ACM 6 (May 1963), p. 194.

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                         PDQ INSERT #1
      A         = NO
 10   A         = A*.7
      I         = A
      IF(I .LE. 6)            GO TO 20
      IF(MOD(I,2) .EQ. 0)     I = I + 1
      K         = NO - I

      DO 15 J = 1,K
      IF(NUM(J).LE.NUM(I+J)) GO TO 15
      MAX       = NUM(   J)
      NUM(   J)= NUM(I+J)
      NUM(I+J)= MAX
 15   CONTINUE
      GO TO 10

 20   K         = NO - 1
      DO 30 J = 1,K
      L         = J
 25   IF(NUM(L).LE.NUM(L+1)) GO TO 30
      MAX       = NUM(L   )
      NUM(L   )= NUM(L+1)
      NUM(L+1)= MAX
      L         = L - 1
      IF(L .GT. 0)            GO TO 25
 30   CONTINUE
      RETURN
C                                                         LISTING  6
      END
      SUBROUTINE SORT
      COMMON NUM(10000),NO
C                                                         BUBBLE SORT
      K         = NO - 1
 5    L         = 1

      DO 10 J = 1,K
      IF(NUM(J).LE.NUM(J+1)) GO TO 10
      MAX       = NUM(J   )
      NUM(J   )= NUM(J+1)
      NUM(J+1)= MAX
      L         = J
 10   CONTINUE
      IF(L .EQ. 1)            RETURN
      K         = L - 1
      GO TO 5
C                                                         LISTING  7
      END
      SUBROUTINE SORT(F)
      COMMON NUM(90000),NO,TIME(7)
C                                                         DISTRIBUTION #1
      A         = NO
      I         = NO
 10   IF(I .LE. 1)            RETURN
      A         = A*F
      I         = A
      IF(I .LT. 1)            I = 1
      K         = NO - I

      DO 20 J = 1,K
      L         = J
 15   IF(NUM(L).LE.NUM(I+L)) GO TO 20
      MAX       = NUM(   L)
      NUM(   L)= NUM(I+L)
      NUM(I+L)= MAX
      L         = L - I
      IF(L .GT. 0)            GO TO 15
 20   CONTINUE
      GO TO 10
C                                                         LISTING  8
      END
 10000       10000       10000 D#1       L8         0        C
                         TIME IN SECONDS
    NO           F       TIME1     TIME2
                        RANDOM    SORTED
 10000        .200       2.93       .36
 10000        .205       1.33       .33
 10000        .210       1.31       .33
 10000        .215       1.40       .34
 10000        .220       1.30       .33
 10000        .225       1.48       .34
 10000        .230       1.41       .34
 10000        .235       1.54       .34
 10000        .240       1.31       .33
 10000        .245       1.37       .39
 10000        .250       2.81       .40
 10000        .255       1.32       .39
 10000        .260       1.22       .39
 10000        .265       1.30       .39
 10000        .270       1.21       .39
 10000        .275       1.36       .39
 10000        .280       1.17       .39
 10000        .285       1.29       .39
 10000        .290       1.61       .39
 10000        .295       1.28       .40
 10000        .300       1.19       .45
 10000        .305       1.29       .44
 10000        .310       1.41       .45
 10000        .315       1.20       .45
 10000        .320       1.13       .44
 10000        .325       1.18       .45
 10000        .330       1.20       .45
 10000        .335       1.35       .44
 10000        .340       1.41       .44
 10000        .345       1.32       .52
 10000        .350       1.56       .50
 10000        .355       1.12       .50
 10000        .360       1.16       .49
 10000        .365       1.13       .50
 10000        .370       1.14       .49
 10000        .375       1.13       .49
 10000        .380       1.13       .50
 10000        .385       1.47       .50
 10000        .390       1.12       .56
 10000        .395       1.21       .55
 10000        .400       1.25       .55
 10000        .405       1.15       .55
 10000        .410       1.12       .55
 10000        .415       1.15       .54
 10000        .420       1.10       .54
 10000        .425       1.10       .54
 10000        .430       1.12       .60
 10000        .435       1.13       .60
 10000        .440       1.13       .60
 10000        .445       1.18       .60
 10000        .450       1.14       .61
 10000        .455       1.12       .60
 10000        .460       1.12       .59
 10000        .465       1.15       .66
 10000        .470       1.15       .66
 10000        .475       1.16       .66
 10000        .480       1.14       .65
 10000        .485       1.17       .65
 10000        .490       1.19       .67
 10000        .495       1.25       .70
 10000        .500       1.46       .71
```

Figure 4. Elapsed Time as a Function of F for Distribution 1 (Listing 8).

Generally, the SHELL sort does not qualify as a merge algorithm, but it can be made to operate as such under definition (1) by selecting F = .5 and the first interval as the largest power of 2 less than NO.

This choice of F turns out to be one of the worst possible. To see why, it may be constructive to consider this procedure as a type of distribution sort. Consider the sequence of 16 numbers, 1, 2, . . ., 16 where the integers represent the position in an array of numbers to be sorted. Take the first interval of comparison, I_1 equal to 8 and F = .5, then the following illustration can be used:

1	2	3	4	5	6	7	8	Pass 1
9	10	11	12	13	14	15	16	$I_1 = 8$

to suggest that the array to be sorted has been divided into eight subsequences represented by the columns. The numbers in each column are to be sorted in ascending order from top to bottom; i.e., the number in position 1 is to be less than or equal to the number in position 9, etc.

The following illustration represents the next pass:

1	2	3	4	Pass 2
5	6	7	8	$I_2 = 4$
9	10	11	12	
13	14	15	16	

Note that: (1) the subsequences are reduced in number in proportion to F, (2) the length of the subsequences is increased in inverse proportion to F, and (3) the elements of a given column are composed alternately of the n and n + $I_1/2$ columns of the previous pass, n = 1, 2, 3, $I_1/2$. Consequently, there is a high degree of order in any column and an element is likely to be close to its sorted position in a subsequence, and an INSERTION sort or BUBBLE sort applied to a column may be expected to operate faster than on a random distribution of the same length. The worst case in this example occurs when the even and odd columns represent disjoint ranges, and the set of numbers from the even columns contains the smaller numbers. Insofar as the median of each column represents the median of the entire distribution, subsequent passes may be expected to produce increasingly well-ordered subsequences representative of the entire sequence; therefore, needing a minimum number of comparisons and exchanges to sort. However, certain difficulties can arise.

In the following example, the array notation of the previous illustration will be retained, but now the integers represent the actual numbers in the array that are to be sorted. Given the sequence 3, 5, 6, 7, 2, 8, 9, 10, 4, 11, 12, 13, 1, 14, 15, 16, let F = .5 and $I_1 = 8$, we have:

3	5	6	7	2	8	9	10	Pass 1
4	11	12	13	1	14	15	16	$I_1 = 8$

All columns but the fifth are in sort; therefore, the 1 and 2 elements are exchanged. This could be called an unfavorable exchange, as it improves the order of the array very little; both the 1 and the 2 should be in the first row. After the exchange, pass 2 can be represented:

3	5	6	7	Pass 2
1	8	9	10	$I_2 = 4$
4	11	12	13	
2	14	15	16	

The columns are well-ordered except column 1. After the sort we have:

1	5	6	7
2	8	9	10
3	11	12	13
4	14	15	16

Column 1 has a median atypical of the array, and most of its elements are far from their final position. It will take a large number of small steps to move them into postion in later passes. The elements in column 1 after row 1 may be thought of as blocking elements because they inhibit efficient distribution of the array elements in the early stages of the sort when an element can proceed toward its destination by large steps. To follow the example further:

1	5	Pass 3	1	5
6	7	$I_3 = 2$	2	7
2	8		3	8
9	10		4	10
3	11		6	11
12	13		9	13
4	14		12	14
15	16		15	16

The last pass with $I_4 = 1$ will require a large number of exchanges to complete the sort. The collection of blocking elements can be prevented by a number of methods, such as sorting up a diagonal from left to right, following a column sort. But most of the comparisons will not result in an exchange due to a high degree of order produced by the previous pass, and those exchanges that do result are bought at the expense of an extra pass. Similar objections may be raised to other modifications to the algorithm to eliminate the blocking elements. Examination of the problem reveals that the accumulation of blocking elements from pass to pass is particularly severe if the rows in the array are divided into an integral number of rows in the next pass; therefore, .25 and .5 are bad values for F. This can be verified readily by rewriting the sort algorithm to include counts of primary exchanges and secondary exchanges on a pass-by-pass basis, and printing these values (as well as their cumulative totals), for selected values of F (Figures 5 and 6). It is interesting to note that the minimum sort times do not correspond to the values of F that result in minimum exchanges, because as F becomes larger, there is an increase in overhead associated with the increase in the number of passes. The elapsed times shown are distorted by the code modification required for the accumulation and printing of the statistics.

A study of the exchange counts suggests that the blocking phenomena come into play whenever there is a series of intervals that have factors in common, and that the problem becomes more severe as F decreases; therefore, the intervals should be relatively prime. Consider the first column of the array on the mth pass. The location of a blocking element in the first column can be expressed as 1 + M*Pm. On the next pass, if that same element is to appear in the same column, it must have a location expressible as 1 + N*Pn, where Pn < Pm. Then N*Pn = M*Pm, and N = Pm, M = Pn if Pm, Pn are relatively prime. But these values for M and N are impossible during the earlier passes of the sort for large arrays and the range of F considered here. To test this hypothesis, sequences of relatively prime numbers corresponding to the geometric sequences associated to values for F (F = .2 to F = .46) were used in the sort algorithm (Listing 9) to conduct elapsed time tests (Figure 7). The tests seem to confirm the hypothesis; however, the use of prime sequences is only a partial solution. A blocking element may be replaced in a column with another element that also serves as a blocking element. This situation is likely to occur when sorting arrays of equal length sorted blocks. Counts of exchanges and comparisons for the prime sequences are given in Figure 8.

PRIME SORTING ALGORITHM

The prime sequence corresponding to F = .3 was selected with a view to minimum combined elapsed times for both random and sorted distributions for the algorithm PRIME SORT (Listing 10). This algorithm also incorporates a change that significantly improves the efficiency of exchanges, especially secondary exchanges.

The improvement in execution speed was largely lost when the data was passed through the CALL list. This illustrates a compilation problem that affects the various algorithms given here to differing extents; the PDQ codes were degraded least by passing data through the CALL list. Efficient compilation is important because the PDQ, PRIME, and DISTRIBUTION sort codes given here derive their speed from compact code that requires a minimum of instructions to execute, and from improved partitioning schemes. Full realization of the potential of the code requires effective register assignment to indices, etc., but variables passed through CALL lists inhibit optimal compilation. Generally, an improvement in performance may be expected when data is passed to a subroutine through COMMON storage (Figure 9).

```
10000      10000      10000 COUNTS            0            0
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .200   2000     1       5435        4515       8000       7451
         .200    400     2       6531       13457       9600      19496
         .200     80     3       7321       39342       9920      46552
         .200     16     4       7720      101590       9984     109354
         .200      3     5       9036      222497       9997     231548
         .200      1     6       5471        9871       9999      14741

      TOTALS---------------------41536------390372------57500-----428772-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .210   2199     1       5305        4187       7901       6871
         .210    440     2       6645       13879       9560      19973
         .210     92     3       7719       29051       9908      36644
         .210     19     4       8125       39808       9981      47903
         .210      4     5       7399       33340       9996      46325
         .210      1     6       6553       15958       9999      22509

      TOTALS---------------------42333------141333------57345-----180225-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .220   2199     1       5185        3850       7801       6391
         .220    483     2       6561       12443       9517      18392
         .220    106     3       7556       24802       9894      32235
         .220     23     4       7941       36819       9977      44732
         .220      5     5       8022       38136       9995      45149
         .220      1     6       7108       24748       9999      31853

      TOTALS---------------------42373------140798------57183-----179752-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .230   2299     1       5078        3409       7701       5877
         .230    528     2       6415       11366       9472      17118
         .230    121     3       7409       23585       9879      30875
         .230     27     4       8019       35224       9973      43210
         .230      6     5       7831       30339       9994      38164
         .230      1     6       7550       41386       9999      48936

      TOTALS---------------------42303------145309------57018-----184180-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .240   2399     1       4927        3079       7601       5335
         .240    575     2       6211       10322       9425      15851
         .240    138     3       7294       20518       9862      27639
         .240     33     4       7672       26512       9967      34256
         .240      7     5       7905       32142       9993      40040
         .240      1     6       7847       41734       9999      49580

      TOTALS---------------------41847------134407------56847-----172701-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .250   2499     1       4830        2738       7501       4845
         .250    624     2       6213        9635       9376      15090
         .250    156     3       5364       14589       9844      20770
         .250     39     4       6937       35514       9961      43405
         .250      9     5       8664      105480       9991     115136
         .250      2     6       8045      223317       9998     231359
         .250      1     7       3590        1819       9999       5408

      TOTALS---------------------44643------393092------66670-----436013-
            F      I  PASS    PRIMARY   SECONDARY    PRIMARY  SECONDARY
                             EXCHANGE    EXCHANGE    COMPARE    COMPARE
         .260   2599     1       4687        2519       7401       4504
         .260    675     2       5935        8384       9325      13590
         .260    175     3       7043       15849       9825      23699
         .260     45     4       7655       25151       9955      32750
         .260     11     5       7897       23527       9989      37407
         .260      3     6       7284       20924       9997      28204
         .260      1     7       5334        6263       9999      11595

      TOTALS---------------------45840------103617------66491-----151739-
```

Figure 5. Counts of Exchanges and Comparisons as a Function of F for the Distribution # 1 Algorithm.

F	PRIMARY EXCHANGE	SECONDARY EXCHANGE	PRIM.+S. EXCHANGE	PRIMARY COMPARE	SECONDARY COMPARE	PRIM.+S. COMPARE	TIME
.200	41134	390372	431906	37500	429772	466272	4.52
.210	42339	141223	183562	57345	180227	237570	1.99
.220	42374	140798	183171	57163	179752	236935	2.08
.230	42302	145310	187611	57018	184180	241198	2.06
.240	41847	134417	176254	56647	172701	229348	1.92
.250	44703	395032	439735	66670	436013	502683	4.40
.260	45640	109517	155457	66491	151739	218230	1.79
.270	45231	107556	152787	66306	149050	215356	1.76
.280	45125	101248	146404	66116	142560	208676	1.71
.290	46849	181379	228228	65922	222388	288310	2.44
.300	47372	86175	133547	75719	129578	205297	1.70
.310	47356	129384	176740	75613	172632	248345	2.05
.320	47518	81923	129441	75300	125242	200542	1.62
.330	47771	91686	139457	75030	135205	210285	1.75
.340	46851	134366	181217	74854	176841	251695	2.08
.350	48763	146655	195426	84620	191968	276588	2.26
.360	43932	77229	120161	84382	121693	206075	1.65
.370	49940	71377	121317	84133	116831	200964	1.61
.380	49698	67314	117012	83077	112568	195645	1.58
.390	51248	55440	106688	93611	102137	195748	1.53
.400	51705	75196	126901	93341	122374	215715	1.72
.410	51545	54414	105959	93058	101340	194398	1.52
.420	51567	54076	105643	92766	101867	194633	1.49
.430	52140	45966	98106	102461	93305	195766	1.50
.440	52694	49156	101850	102151	97004	199155	1.54
.450	52979	47114	99993	101825	95040	196865	1.52
.460	52879	46437	99316	101490	94170	195660	1.49
.470	53917	40506	94423	111140	89195	200335	1.51
.480	53998	41897	95895	110778	90526	201304	1.52
.490	54019	41568	95587	110402	90220	200622	1.56
.500	57933	86443	144376	120909	138791	259790	2.02
.510	55314	38397	93711	119601	88099	207700	1.55
.520	55790	33114	88904	129175	83315	212490	1.52
.530	55279	29921	85200	128733	79624	208357	1.41
.540	55234	30272	85506	128270	79905	208175	1.50
.550	55232	26961	82193	137787	76618	214405	1.56
.560	55672	26324	81996	137284	76395	213679	1.58
.570	57067	26437	83504	146755	77873	224629	1.60
.580	56458	29623	86081	146202	80390	226592	1.64
.590	57741	20546	78287	155622	72690	228312	1.59
.600	56830	20237	77067	155015	71339	226354	1.59
.610	57585	18012	75597	164371	69952	234323	1.64
.620	57610	24339	81949	163696	76398	240104	1.73
.630	56762	23336	80098	172985	74279	247264	1.73
.640	57142	15616	72758	182234	67060	249294	1.70
.650	56516	16366	72982	181442	67088	248530	1.69
.660	56863	13103	69966	190604	64040	254644	1.72
.670	58470	11772	70242	199770	64130	263900	1.73
.680	58212	10576	68788	208765	62663	271428	1.84
.690	58525	10901	69426	207759	62125	269884	1.82
.700	57044	10150	67094	216687	60625	277312	1.90
.710	59227	6386	65613	225535	60110	285645	1.96
.720	57117	7339	65116	234303	58532	292885	1.92
.730	50044	8178	56222	252980	59675	312655	2.05
.740	57850	6939	64789	261558	58290	319838	2.11
.750	57698	5916	63594	270020	56934	326954	2.12
.760	58102	6159	64261	288355	57597	345952	2.19
.770	57237	4958	62195	296546	55446	351992	2.24
.780	57337	4410	62567	314569	55556	370125	2.35
.790	57292	3494	60786	332408	53800	386208	2.47
.800	57691	3177	60868	350026	53734	403760	2.52

Figure 6. Cumulative Counts of Exchanges and Comparisons as a Function of F for the Distribution #1 Algorithm.

```
      SUBROUTINE SORT(NUM,NO,N)
      DIMENSION NUM(NO),INK(16,27)

      DATA((INK(I,J),I=1,16),J=1,19)    /1,5,23,127,631,3121,15629,78121,  F= .20
     *390625,7*9,                         1,5,23,107,509,2447,11657,55529,     .21
     *264390,7*9,                         1,5,23, 97,431,1933, 8821,40093,     .22
     *182229,7*9,                         1,4,19, 93,359,1553, 6761,29363,     .23
     *127696,7*9,                         1,4,17, 71,307,1259, 5237,21803,     .24
     * 90847,378329,6*9,                  1,4,17, 61,257,1021, 4093,16381,     .25
     * 65537,262144,6*9,                  1,4,13, 59,223, 839, 3229,12451,     .26
     * 47881,184179,6*9,                  1,4,13, 53,191, 701, 2579, 9551,     .27
     * 35407,131137,6*9,                  1,4,13, 47,153, 577, 2081, 7411,     .28
     * 25479, 94331,337613,5*9,           1,3,13, 41,139, 487, 1693, 5801,     .29
     * 19991, 68927,237695,5*9,           1,3,11, 37,127, 409, 1373, 4567,     .30
     * 15241, 50921,169351,5*9,           1,3,11, 31,107, 349, 1129, 3637,     .31
     * 11731, 37831,122007,5*9,           1,3,11, 31, 97, 293,  929, 2909,     .32
     *  9091, 28429, 89817,277556,4*9,    1,3,11, 29, 83, 257,  773, 2347,     .33
     *  7109, 21557, 65293,197835,4*9,    1,3, 7, 23, 73, 223,  647, 1901,     .34
     *  5591, 16477, 48437,142473,4*9,    1,3, 7, 23, 67, 191,  547, 1553,     .35
     *  4441, 12589, 36251,103574,4*9,    1,3, 7, 19, 59, 167,  457, 1277,     .36
     *3547,9851,27361,75979,211043,3*9,   1,3, 7, 19, 53, 139,  389, 1051,     .37
     *2851,7699,20789,56207,151908,3*9,   1,3, 7, 17, 47, 127,  331,  877,     .38
     *2297,6053,15923,41911,110395,3*9/
      DATA((INK(I,J),I=1,16),J=20,27)/1,3,7,17,43,113,283,727,1867,4793,       .39
     *12201,31489,80761,207090,2*9,    1,3,7,17,37, 97,241,613,1523,3821,      .40
     * 9539,23833,59611,149012,2*9,    1,2,7,13,37, 83,211,509,1249,3049,      .41
     *7451,18169,44293,108096, 2*9,    1,2,7,13,31, 79,181,433,1033,2459,      .42
     *5857,13933,33191,79031,188152,9,1,2,5,13,29, 67,157,367, 857,1993,       .43
     *4621,10753,25031,58199,135345,9,1,2,5,13,29, 61,137,313, 709,1619,       .44
     *3677,8353,19001,43159,98101,222951,1,2,5,11,23,53,113,269,593,1321       .45
     *,2939,6529,14503,32233,71633,159152,1,2,5,11,23,47,107,229,499,          .46
     *1087,2357,5119,11149,24223,52639,114454/

      DO 5 IN = 1,16
      IF(NO .LE. INK(IN,N))   GO TO 10
    5 CONTINUE

   10 IN       = IN - 1
      IF(IN .LE. 0)           RETURN
      I        = INK(IN,N)
      K        = NO - I

      DO 20 J  = 1,K
      L        = J
   15 IF(NUM(L).LE.NUM(I+L))  GO TO 20
      MAX      = NUM(  L)
      NUM(  L) = NUM(I+L)
      NUM(I+L) = MAX
      L        = L - I
      IF(L .GT. 0)            GO TO 15
   20 CONTINUE
      GO TO 10
C                                                               LISTING 9
      END
```

```
10000          1000 PRIME   L9          0            0
```

		TIME IN SECONDS	
NO	F	TIME1 RANDOM	TIME2 SORTED
10000	.200	1.41	.38
10000	.210	1.37	.39
10000	.220	1.41	.40
10000	.230	1.33	.42
10000	.240	1.36	.43
10000	.250	1.29	.44
10000	.260	1.39	.45
10000	.270	1.25	.46
10000	.280	1.23	.48
10000	.290	1.24	.49
10000	.300	1.23	.50
10000	.310	1.24	.51
10000	.320	1.22	.53
10000	.330	1.24	.54
10000	.340	1.23	.56
10000	.350	1.21	.57
10000	.360	1.20	.58
10000	.370	1.18	.60
10000	.380	1.20	.62
10000	.390	1.20	.64
10000	.400	1.23	.64
10000	.410	1.21	.67
10000	.420	1.24	.69
10000	.430	1.23	.70
10000	.440	1.26	.73
10000	.450	1.24	.74
10000	.460	1.28	.76

Figure 7. Elapsed Time for Prime Sequences, F = .2 to F = .46.

F	PRIMARY EXCHANGE	SECONDARY EXCHANGE	PRIM.+S. EXCHANGE	PRIMARY COMPARE	SECONDARY COMPARE	PRIM.+S. COMPARE	TIME
.200	41462	147881	189343	56092	185639	241731	2.06
.210	42016	139874	181890	56908	178340	235248	1.99
.220	42572	143802	186374	56689	183067	241756	2.02
.230	43221	125396	168617	61220	164868	226088	1.88
.240	44256	127638	171894	63104	167525	230629	1.91
.250	44676	112908	157584	64546	153442	217988	1.79
.260	45485	128351	173836	65632	169789	235421	1.95
.270	45620	101598	147216	66907	143503	210410	1.72
.280	45771	95038	140809	69703	137030	206733	1.69
.290	45559	93253	139812	71822	135500	207322	1.71
.300	47249	88133	135381	73472	130919	204391	1.66
.310	47959	87356	135315	74732	131027	205759	1.66
.320	48197	82176	130373	76635	126432	203067	1.63
.330	49022	80827	129849	79387	125723	205110	1.68
.340	49553	77781	127334	81531	122710	204241	1.62
.350	49793	70060	119853	83167	115293	198460	1.56
.360	51400	67542	117942	84612	113562	198174	1.57
.370	49707	59500	109207	87788	104974	192762	1.51
.380	50391	59552	109943	90240	105303	195543	1.51
.390	51120	55681	106801	92146	101906	194052	1.51
.400	51650	59045	110695	94101	106178	200279	1.55
.410	51510	51370	102880	97388	98356	195744	1.50
.420	52509	52483	104992	99904	100123	200027	1.53
.430	53395	48715	102110	101888	97187	199075	1.52
.440	53763	49395	103158	105081	98489	203570	1.57
.450	53747	43001	96748	108141	91929	200070	1.49
.460	55065	44032	99097	110513	93758	204281	1.53

Figure 8. Cumulative Counts of Exchanges and Comparisons for Prime Sequences, F = .2 to F = .46.

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO),INK(11)
C                          IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
C                                                                PRIME SORT
      DATA INK /1,3,11,37,127,409,1373,4567,15241,50821,169351/          F = .30
      DO 5 IN = 1,11
      IF(NO .LE. INK(IN))     GO TO 10
 5    CONTINUE
 10   IN       = IN - 1
      IF(IN  .LE. 0)          RETURN
      I        = INK(IN)
      K        = NO - I
      DO 20 J = 1,K
      L        = J
      NUM2     = NUM(I+L)
 15   NUM1     = NUM(  L)
      IF(NUM1 .LE. NUM2)      GO TO 20
      NUM(  L)= NUM2
      NUM(I+L)= NUM1
      L        = L - I
      IF(L .GT. 0)            GO TO 15
 20   CONTINUE
      GO TO 10
C ----                                                        LISTING 10
      END
```

Assembly language coding can minimize these problems, and candidates for that purpose are given in Listings 11 and 12. An algorithm suitable for in-line code is given in Listing 13. The effect of a prime sequence is approximated by making all intervals odd, etc.

Expected sort behavior is indicated by the expression for the number of comparisons required by the PDQ sort. The intervals of comparison are given approximately by the sequence:

$$F*NO,\ F^2*NO,\ F^3*NO,\ \ldots,\ F^n*NO$$

such that $F^n*NO = 1$. Then $n = -(\text{Log NO})/(\text{Log F})$. The number of comparisons for n passes are:

$$K = (NO-F*NO) + (NO-F^2*NO) + \ldots + (NO-F^n*NO)$$

$$K = n*NO - (F + F^2 + \ldots + F^n)*NO$$

$$K = -\frac{\text{Log NO}}{\text{Log F}} *NO - (F + F^2 + \ldots + F^n)*NO$$

This is also the expression for the primary comparisons of the DISTRIBUTION sort, but with a smaller value for F.

```
  10000        1000  D#2    L13              0              0
                                        TIME  IN  SECONDS
                  NO           F        TIME1         TIME2
                                       RANDOM        SORTED
               10000        .200         1.24           .33
               10000        .205         1.22           .33
               10000        .210         1.16           .33
               10000        .215         1.25           .33
               10000        .220         1.20           .33
               10000        .225         1.37           .34
               10000        .230         1.20           .33
               10000        .235         1.21           .33
               10000        .240         1.39           .33
               10000        .245         1.07           .39
               10000        .250         1.11           .39
               10000        .255         1.05           .39
               10000        .260         1.06           .39
               10000        .265         1.07           .39
               10000        .270         1.21           .38
               10000        .275         1.16           .39
               10000        .280         1.06           .38
               10000        .285         1.05           .39
               10000        .290         1.08           .38
               10000        .295         1.06           .38
               10000        .300         1.02           .44
               10000        .305         1.10           .44
               10000        .310         1.00           .44
               10000        .315          .97           .44
               10000        .320         1.08           .44
               10000        .325         1.10           .44
               10000        .330         1.08           .44
               10000        .335         1.14           .43
               10000        .340         1.04           .43
               10000        .345          .98           .49
               10000        .350          .95           .49
               10000        .355         1.00           .49
               10000        .360         1.00           .49
               10000        .365          .95           .49
               10000        .370          .97           .49
               10000        .375          .97           .49
               10000        .380          .96           .48
               10000        .385          .95           .49
               10000        .390          .98           .54
               10000        .395         1.02           .55
               10000        .400          .97           .54
               10000        .405          .94           .54
               10000        .410          .94           .54
               10000        .415          .97           .54
               10000        .420          .96           .54
               10000        .425          .94           .54
               10000        .430          .97           .60
               10000        .435          .95           .59
               10000        .440          .95           .59
               10000        .445          .94           .59
               10000        .450          .94           .59
               10000        .455          .96           .59
               10000        .460         1.00           .59
               10000        .465          .95           .65
               10000        .470          .98           .64
               10000        .475          .96           .64
               10000        .480          .95           .64
               10000        .485          .94           .64
               10000        .490          .96           .64
               10000        .495          .99           .70
               10000        .500         1.13           .70
```

Figure 9. Elapsed Time as a Function of F for Distribution 2 (Listing 13).

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                          DISTRIBUTION #3
C                         IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
      A         = NO
      I         = NO
 10   IF(I .LE. 1)          RETURN
      A         = A*.381
      I         = A
      IF(MOD(I,2) .EQ. 0)   I = I + 1
      IF(MOD(I,9) .EQ. 0)   I = I + 2
      K         = NO - I

      J         = 0
 15   J         = J + 1
      IF(J .GT. K)          GO TO 10
      L         = J
      NUM2      = NUM(I+L)
 20   NUM1      = NUM(  L)
      IF(NUM1 .LE. NUM2)    GO TO 15
      NUM(  L)= NUM2
      NUM(I+L)= NUM1
      IF(L .LE. I)          GO TC 15
      L         = L - I
      GO TO 20
C                                                               LISTING 11
      END
```

```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)

C                                                          DISTRIBUTION #4
C                         IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
      A         = NO
      I         = NO
 10   IF(I .LE. 1)          RETURN
      A         = A*.381
      I         = A
      IF(MOD(I,2) .EQ. 0)   I = I + 1
      IF(MOD(I,9) .EQ. 0)   I = I + 2
      K         = NO - I

      J         = 0
 15   J         = J + 1
      IF(J .GT. K)          GO TO 10
      NUM1      = NUM(  J)
      NUM2      = NUM(I+J)
      IF(NUM1 .LE. NUM2)    GO TO 15
      L         = J
 20   NUM(I+L)= NUM1
      IF(L .LE. I)          GO TO 25
      L         = L - I
      NUM1      = NUM(  L)
      IF(NUM1 .GT. NUM2)    GO TO 20
      L         = L + I
 25   NUM(  L)= NUM2
      GO TO 15
C                                                               LISTING 12
      END
      SUBROUTINE SORT(F)
      COMMON NUM(90000),NO,TIME(7)
C                                                        DISTRIBUTION #2
      A          = NO
      I          = NO
 10   IF(I .LE. 1)              RETURN
      A          = A*F
      I          = A
      IF(MOD(I,2) .EQ. 0)       I = I + 1
      IF(MOD(I,9) .EQ. 0)       I = I + 2
      K          = NO - I

      DO 20 J = 1,K
      L          = J
      NUM2       = NUM(I+L)
 15   NUM1       = NUM(  L)
      IF(NUM1 .LE. NUM2)        GO TO 20
      NUM(  L)=  NUM2
      NUM(I+L)=  NUM1
      L          = L - I
      IF(L .GT. 0)              GO TO 15
 20   CONTINUE
      GO TO 10
C                                                        LISTING 13
      END
```

For each primary comparison, there is a chance of a primary exchange which will be followed generally by a secondary comparison, etc. Figure 6 indicates that secondary exchanges will be substantial for useful values of F; therefore, the expression for the comparisons of the DISTRIBUTION or PRIME sort will be that for PDQ sort, plus a series of terms involving probabilities which represent the various orders of exchanges (primary, secondary, etc.). Sort times for large NO may be expected to be proportional to Log NO for the PDQ sort, and to increase faster than a Log NO rate for PRIME sort.

PERFORMANCE EVALUATION

For a comparative evaluation of sort performance, some establised sort algorithms published in the COMMUNICATIONS of the ACM[4-7] were used. These algorithms were adapted for the sake of uniform notation and style. In addition, the subroutine SIFTUP of TREESORT3 was coded in-line to reduce the substantial overhead involved in the frequent calls to this procedure. The sort designated SINGLETON is one of the faster, more stable members of the QUICKSORT family.

[4]Singleton, Richard C. An efficient algorithm for sorting with minimal storage. Communications ACM 12 (March 1969), p. 185.

[5]Loeser, Rudolf. Some performance tests of "QUICKSORT" and decendants. Communications ACM 17 (March 1974), p. 143.

[6]Boothroyd, J. Algorithm 201, SHELLSORT. Communications ACM 6, 8 (August 1963), p. 445.

[7]Floyd, R. W. Algorithm 245, TREESORT3. Communications ACM 7, 12 (December 1964), p. 701.

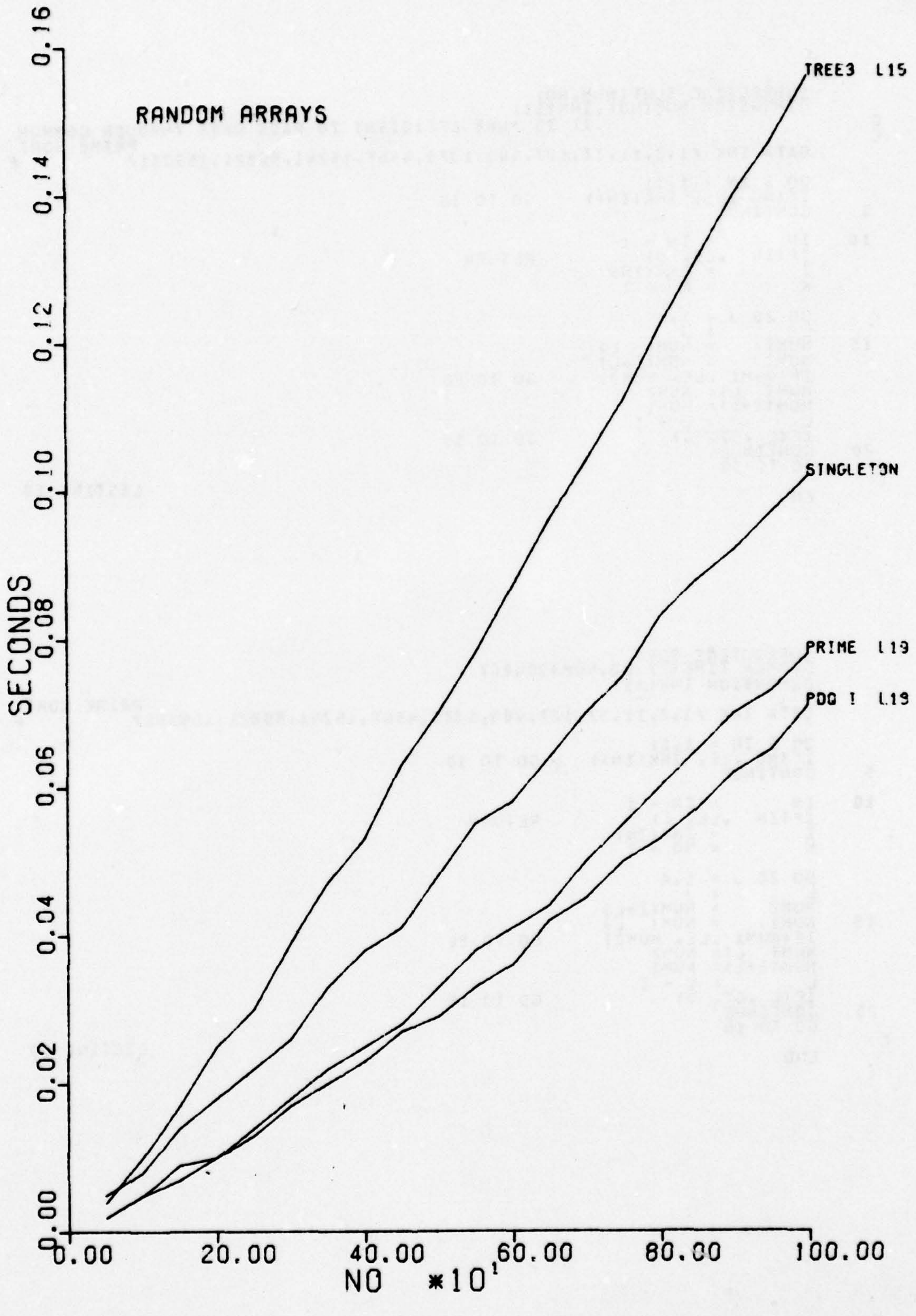
An attempt to evaluate the efficiency of a procedure by frequency counts of critical parameters is not entirely satisfactory, even for differing versions of that procedure on the same machine. The particular form of a procedure is very important. Machine independent comparisons between differing procedures are even more difficult. Some of the results were omitted from the graphical sort performance data when interference between curves obscured the comparisons. Each curve has a label that refers to the listing of the code used in generating the data. (e.g., PDQ I L18 means PDQ INSERT #2, Listing 18.) The test results indicate that the PRIME and DISTRIBUTION sorts compare favorably overall to the QUICKSORT; and in the case of random distributions, the results increasingly favor the PDQ, PRIME, and DISTRIBUTION sorts as the array size decreases. The DISTRIBUTION sort is a compact and efficient sort suitable for in-line code applications because it is generally understandable; therefore, it may be modified easily for particular uses.

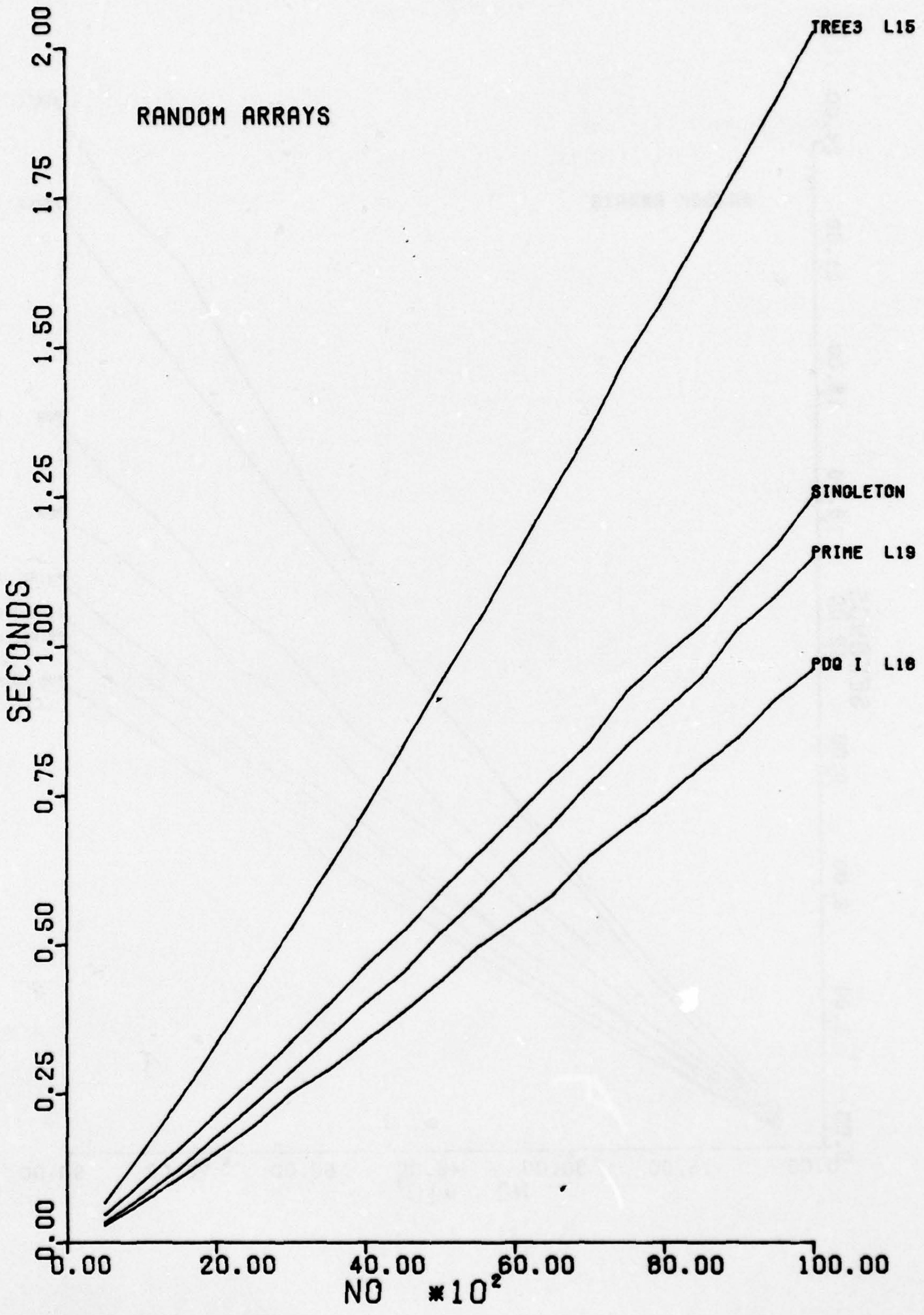
```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C           ADAPTATION OF SHELLSORT (ACM ALGORITHM 201 BY J. BOOTHROYD)
C        COMMUNICATIONS OF THE ACM - VOLUME 17 / NUMBER 3 / MARCH, 1974
      I         = 1
  5   I         = I + I
      IF(I .LE. NO)           GO TO 5
      I         = I - 1
 10   I         = I/2
      IF(I .LE. 0)            RETURN
      K         = NO - I
      DO 20 J = 1,K
      L         = J + I
 15   L         = L - I
      IF(L .LE. 0)            GO TO 20
      M         = L + I
      IF(NUM(L) .LE. NUM(M))  GO TO 20
      MAX       = NUM(L)
      NUM(L)    = NUM(M)
      NUM(M)    = MAX
      GO TO 15
 20   CONTINUE
      GO TO 10
C                                                              LISTING 14
      END
```

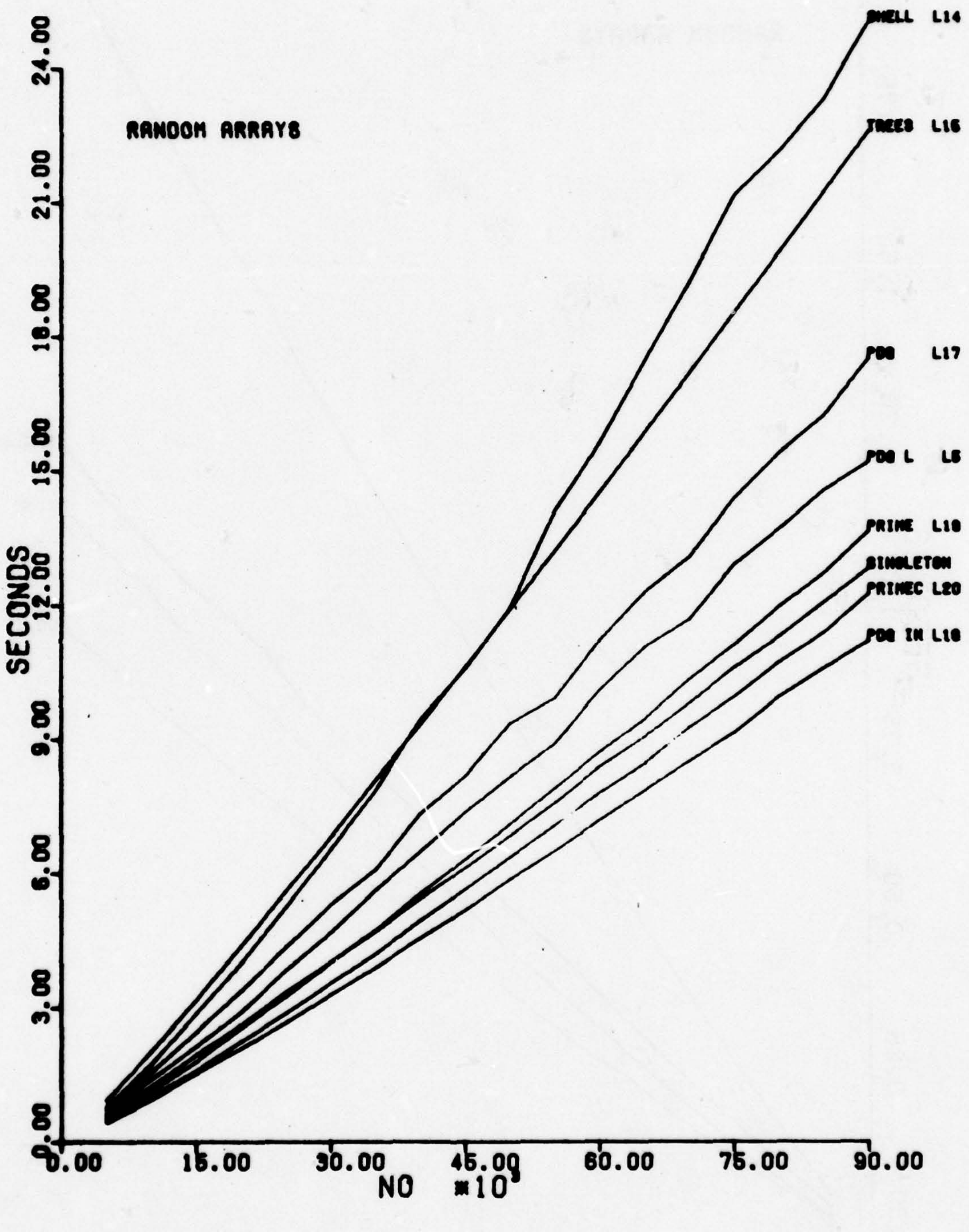
```
      SUBROUTINE SORT(NUM,NO)
C        COMMUNICATIONS OF THE ACM - VOLUME 17 / NUMBER 3 / MARCH, 1974
C             ADAPTATION OF TREESORT3 (ACM ALGORITHM 245 BY R. W. FLOYD)
      DIMENSION NUM(NO)
      N1        = NO
      L         = NO/2 + 1
 10   L         = L - 1
      IF(L .LE. 1)               GO TO 35
C                                SUBROUTINE SIFTUP
      I         = L
      NNNN      = NUM(I)
 20   J         = I + I
      IF(J .GT. N1)              GO TO 30
      IF(J .EQ. N1)              GO TO 25
      IF(NUM(J).LT.NUM(J+1))     J = J + 1
 25   IF(NUM(J).LE.NNNN     )    GO TO 30
      NUM(I)    = NUM(J)
      I         = J
      GO TO 20
 30   NUM(I)    = NNNN

      GO TO 10
 35   L         = N1 + 1
 40   L         = L - 1
      IF(L .LE. 1)               RETURN
C                                SUBROUTINE SIFTUP
      I         = 1
      NNNN      = NUM(1)
 45   J         = I + I
      IF(J .GT. L)               GO TO 60
      IF(J .EQ. L)               GO TO 55
      IF(NUM(J).LT.NUM(J+1))     J = J + 1
 55   IF(NUM(J).LE.NNNN     )    GO TO 60
      NUM(I)    = NUM(J)
      I         = J
      GO TO 45
 60   NUM(I)    = NNNN

      NNNN      = NUM(1)
      NUM(1)    = NUM(L)
      NUM(L)    = NNNN
      GO TO 40
C                                                              LISTING 15
      END
      SUBROUTINE SORT(NUM,NO)
C                                    (ACM ALGORITHM 347) BY RICHARD C. SINGLETON
C        . COMMUNICATIONS OF THE ACM - VOLUME 12 / NUMBER 3 / MARCH, 1969
      DIMENSION IL(17),IU(17),NUM(NO)
      M          = 1
      I          = 1
      J          = NO
  5   IF(I .GE. J)              GO TO 70
 10   K          = I
      IJ         = (J+I)/2
      NT         = NUM(IJ)
      IF(NUM(I)  .LE. NT)       GO TO 20
      NUM(IJ) =  NUM(I )
      NUM(I ) =  NT
      NT         = NUM(IJ)
 20   L          = J
      IF(NUM(J)  .GE. NT)       GO TO 40
      NUM(IJ) =  NUM(J )
      NUM(J ) =  NT
      NT         = NUM(IJ)
      IF(NUM(I)  .LE. NT)       GO TO 40
      NUM(IJ) =  NUM(I )
      NUM(I ) =  NT
      NT         = NUM(IJ)
                                GO TO 40
 30   NUM(L ) =  NUM(K )
      NUM(K ) =  NTT
 40   L          = L - 1
      IF(NUM(L)  .GT. NT)       GO TO 40
      NTT        = NUM(L )
 50   K          = K + 1
      IF(NUM(K)  .LT. NT)       GO TO 50
      IF(K .LE. L)              GO TO 30
      IF(L-I  .LE. J-K)         GO TO 60
      IL(M)      = I
      IU(M)        = L
      I          = K
      M          = M + 1
                                GO TO 80
 60   IL(M)      = K
      IU(M)      = J
      J          = L
      M          = M + 1
                                GO TO 80
 70   M          = M - 1
      IF(M .EQ. 0)              RETURN
      I          = IL(M)
      J          = IU(M)
 80   IF(J-I .GE. 1)            GO TO 10
      IF(I    .EQ. 1)           GO TO 5
      I          = I - 1
 90   I          = I + 1
      IF(I .EQ. J)              GO TO 70
      NT         = NUM(I+1)
      IF(NUM(I)  .LE. NT)       GO TO 90
      K          = I
 100  NUM(K+1)=  NUM(K )
      K          = K - 1
      IF(NT .LT. NUM(K))        GO TO 100
      NUM(K+1)=  NT
                                GO TO 90
C                                                                  LISTING 16
      END
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                                 PDQ #2
      A          = NO
 10   A          = A*.8
      I          = A
      IF(I  .LE.  0)            RETURN
      K          = NO - I

      DO 15  J   = 1,K
      NUM1       = NUM(   J)
      NUM2       = NUM(I+J)
      IF(NUM1 .LT. NUM2)        GO TO 15
      NUM(   J)= NUM2
      NUM(I+J)= NUM1
 15   CONTINUE
      GO TO 10
C                                                             LISTING 17
      END
```

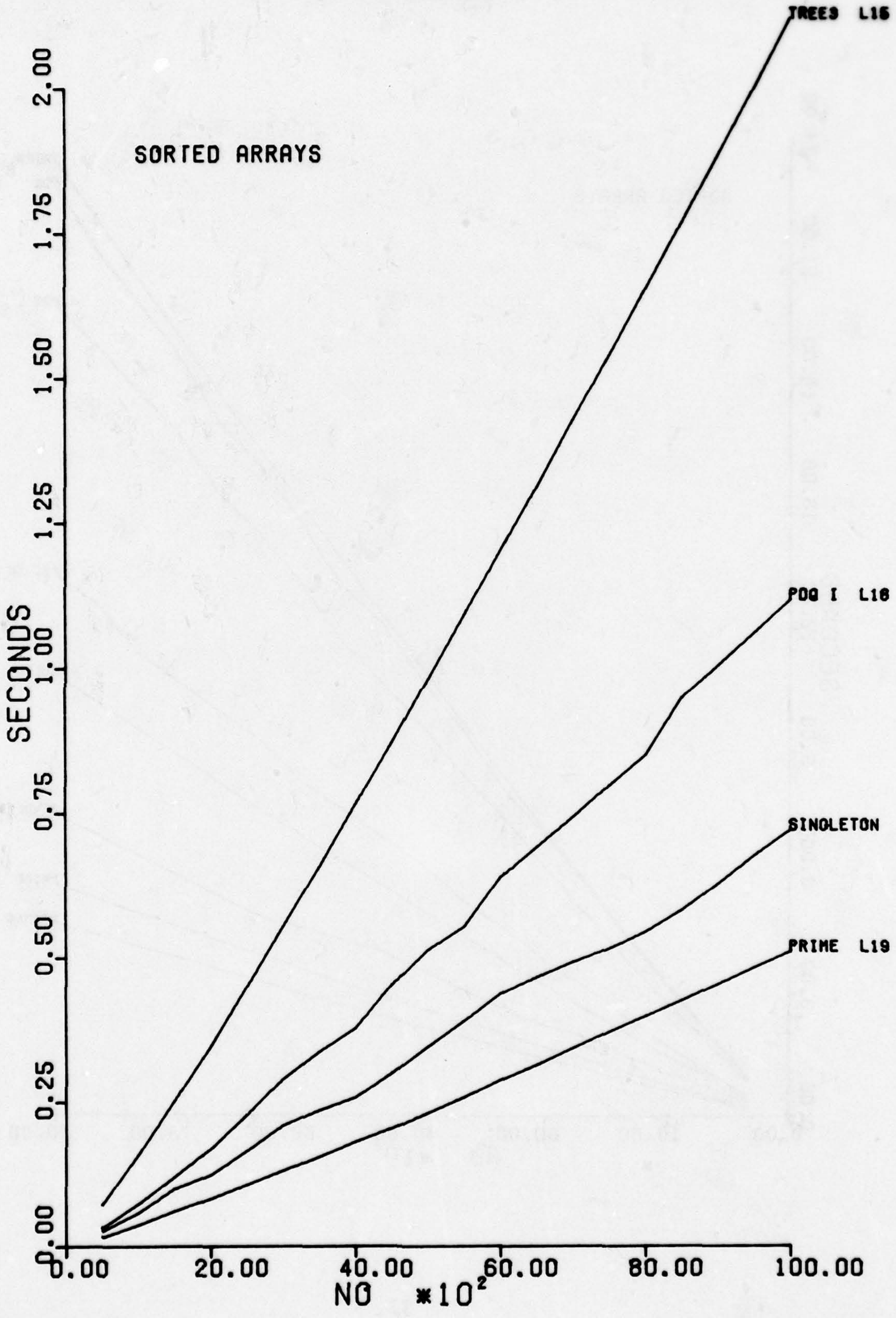
```
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO)
C                                                          PDQ INSERT #2
      A          = NO
      I          = NO
 10   IF(I .LE.  5)             GO TO 30
      A          = A*.715
      I          = A
      IF( I.GT.9 .AND. MOD(I,2).EQ.0)  I = I + 1
      K          = NO - I

      DO 15 J    = 1,K
      NUM1       = NUM(   J)
      NUM2       = NUM(I+J)
      IF(NUM1  .LT.  NUM2)      GO TO 15
      NUM(   J)= NUM2
      NUM(I+J)= NUM1
 15   CONTINUE
      GO TO 10

 30   K          = NO - 1
      DO 40 J    = 1,K
      L          = J
      NUM2       = NUM(L+1)
 35   NUM1       = NUM(L   )
      IF(NUM1  .LE.  NUM2)      GO TO 40
      NUM(L  )= NUM2
      NUM(L+1)= NUM1
      L          = L - 1
      IF(L  .GT.  0)            GO TO 35
 40   CONTINUE
      RETURN
C                               ENHANCEMENTS:
C                               ALTERNATE COMPARISONS
C                IF THE ARRAY NUM IS IMMEDIATELY PRECEEDED BY THE SMALLEST
C                     NUMBER EXPRESSABLE ON THE MACHINE, THAN THE STATEMENT
C                          IF(L.GT.0) GO TO 35 CAN BE REPLACED BY:  GO TO 35
C                                                             LISTING 18
      END
      SUBROUTINE SORT(NUM,NO)
      DIMENSION NUM(NO),INK(11)
C                             IT IS MORE EFFICIENT TO PASS DATA THROUGH COMMON
C                                                                     PRIME SORT
      DATA INK /1,3,11,37,127,409,1373,4567,15241,50821,169351/            F = .30
      DO 5 IN = 1,11
      IF(NO .LE. INK(IN))     GO TO 10
 5    CONTINUE
 10   IN       = IN - 1
      IF(IN  .LE. 0)          RETURN
      I        = INK(IN)
      K        = NO - I
      DO 20 J = 1,K
      L        = J
 15   NUM1     = NUM(  L)
      NUM2     = NUM(I+L)
      IF(NUM1 .LE. NUM2)      GO TO 20
      NUM(  L)= NUM2
      NUM(I+L)= NUM1
      L        = L - I
      IF(L .GT. 0)            GO TO 15
 20   CONTINUE
      GO TO 10
C                                                                     LISTING 19
      END
```

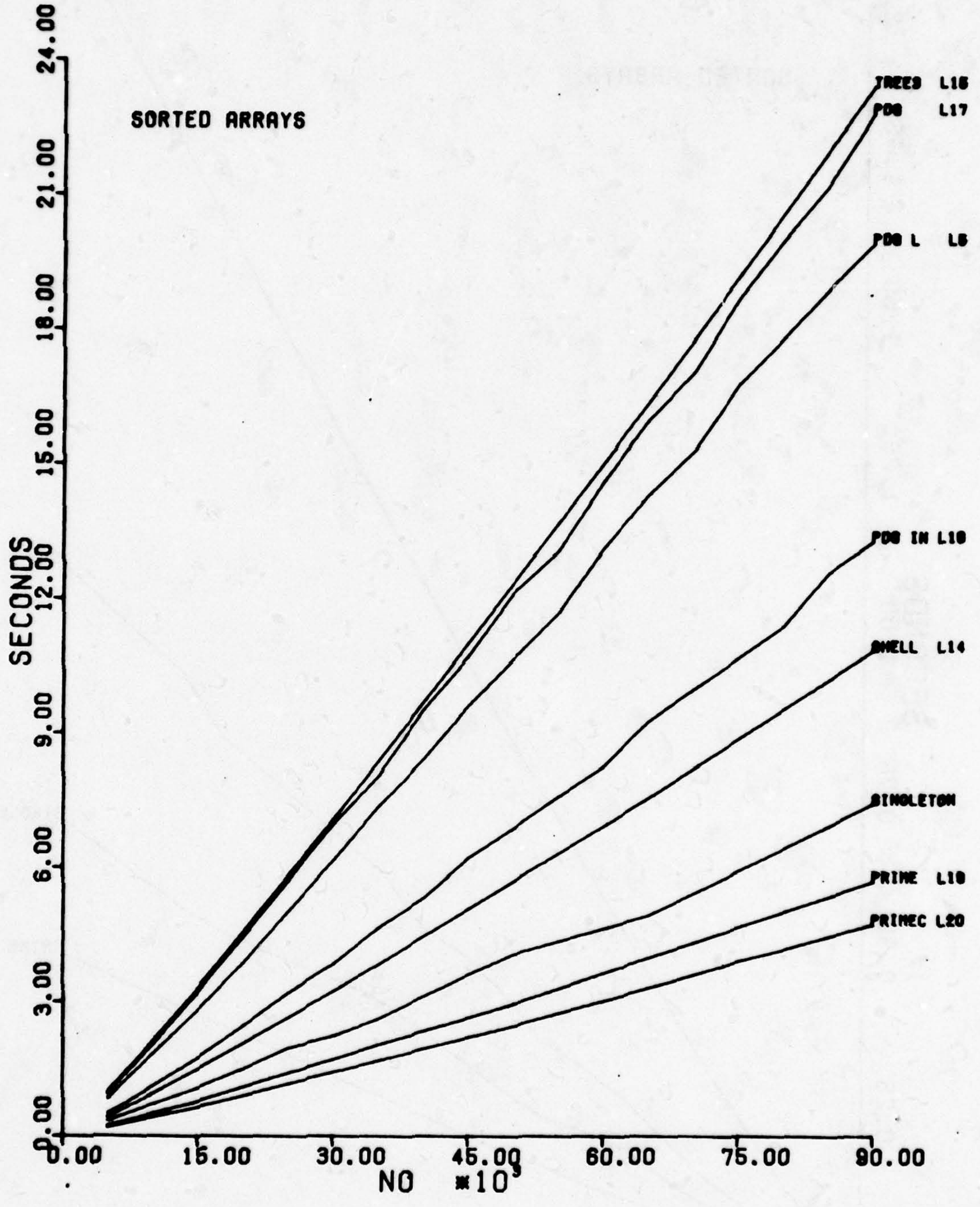
```
      SUBROUTINE SORT
      COMMON TIME(7),NO,NUM(90000)
      DIMENSION INK(11)
C                                                                     PRIME SORT
      DATA INK /1,3,11,37,127,409,1373,4567,15241,50821,169351/            F = .30
      DO 5 IN = 1,11
      IF(NO .LE. INK(IN))     GO TO 10
 5    CONTINUE
 10   IN       = IN - 1
      IF(IN  .LE. 0)          RETURN
      I        = INK(IN)
      K        = NO - I
      DO 20 J = 1,K
      L        = J
      NUM2     = NUM(I+L)
 15   NUM1     = NUM(  L)
      IF(NUM1 .LE. NUM2)      GO TO 20
      NUM(  L)= NUM2
      NUM(I+L)= NUM1
      L        = L - I
      IF(L .GT. 0)            GO TO 15
 20   CONTINUE
      GO TO 10
C                                                                     LISTING 20
      END
```

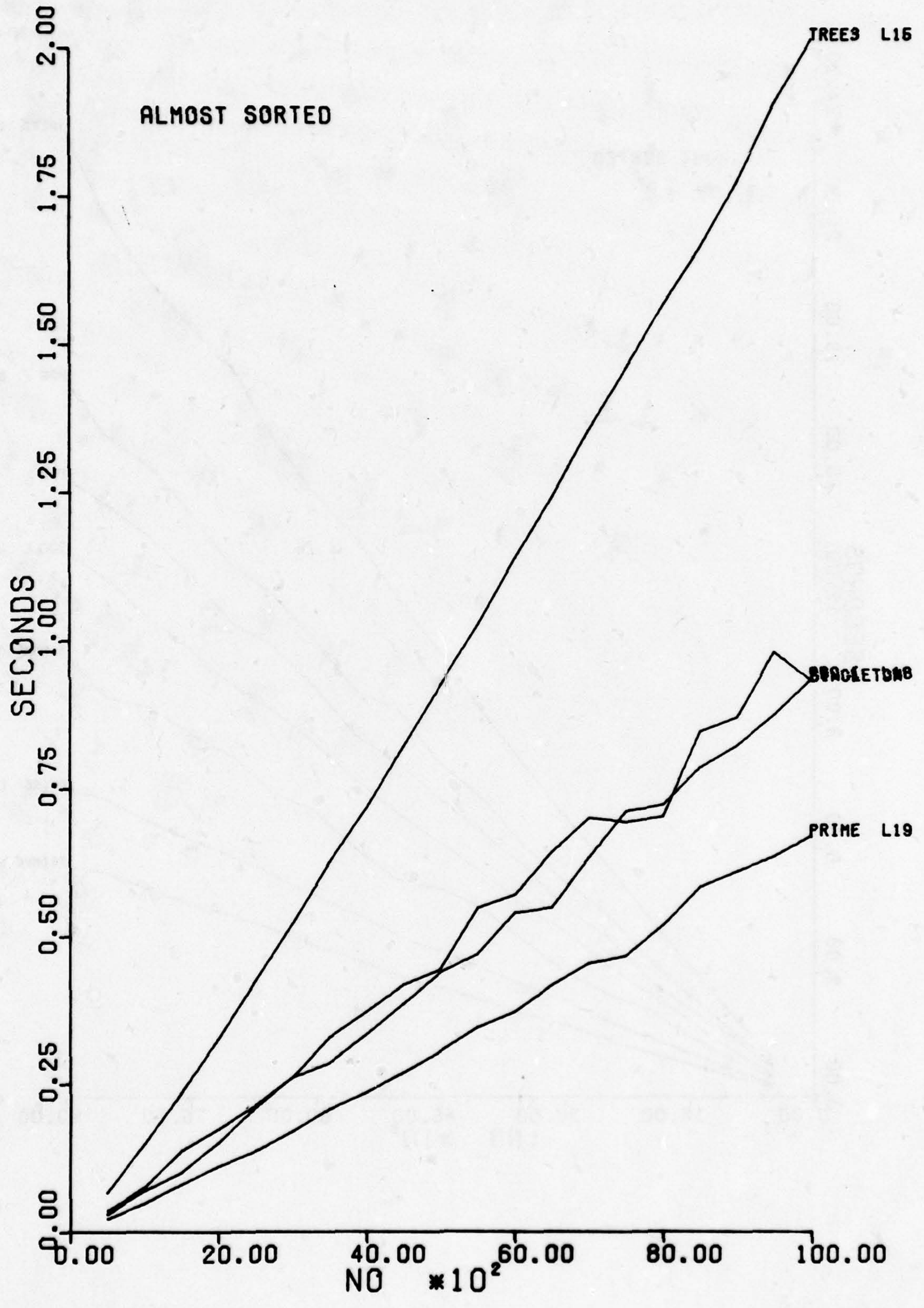

RANDOM ARRAYS
TREE3 L15
SINGLETON
PRIME L19
PDQ ! L19
SECONDS
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0.00
20.00
40.00
60.00
80.00
100.00
NO $*10^1$

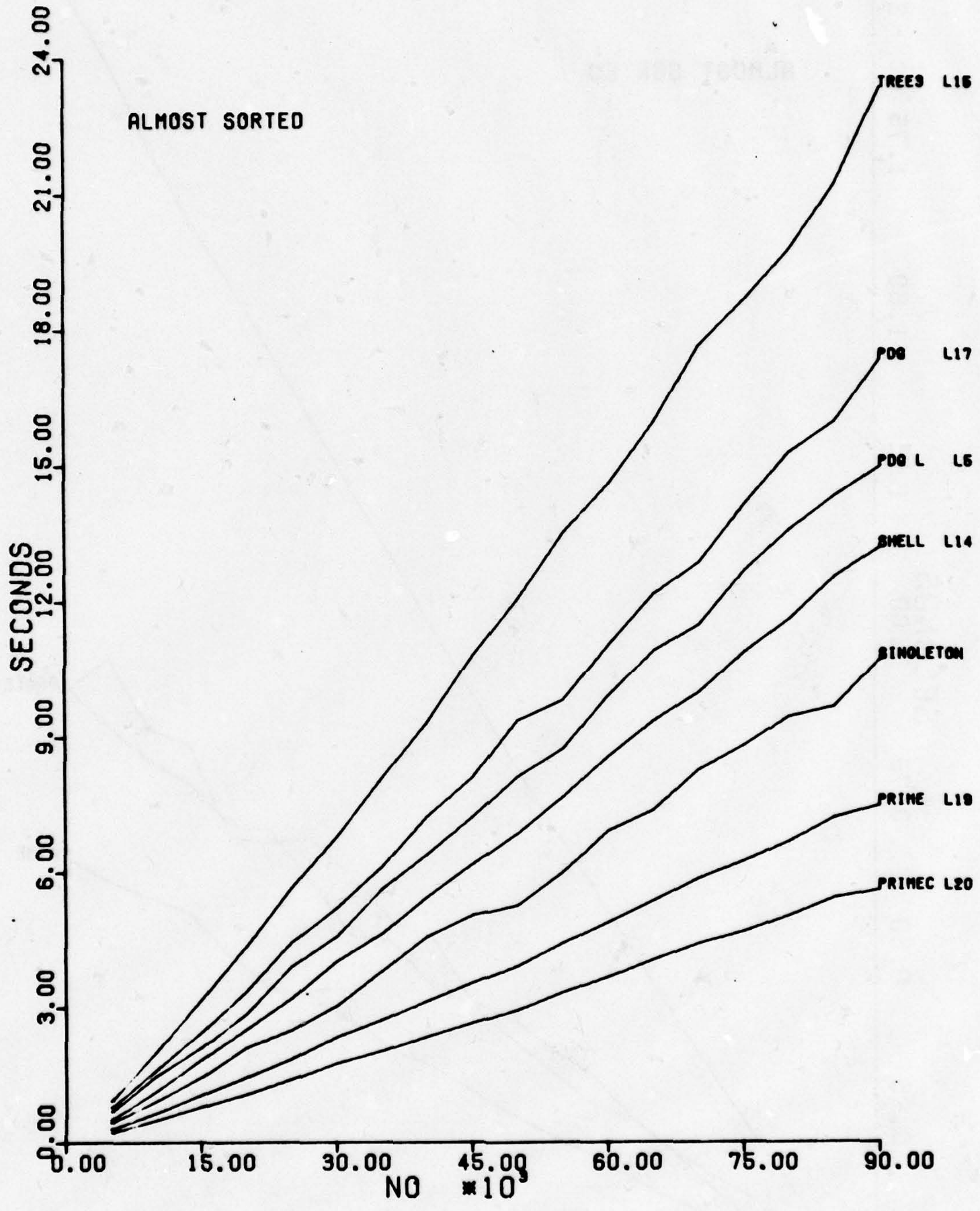

RANDOM ARRAYS
TREE3 L15
SINGLETON
PRIME L19
PDQ I L18
SECONDS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00
NO $*10^2$


RANDOM ARRAYS
SECONDS
NO *10
0.00
3.00
6.00
9.00
12.00
15.00
18.00
21.00
24.00
15.00
30.00
45.00
60.00
75.00
90.00
SHELL L14
TREES L15
PDS L17
PDS L L5
PRIME L19
SINGLETON
PRIMEC L20
PDS IN L18

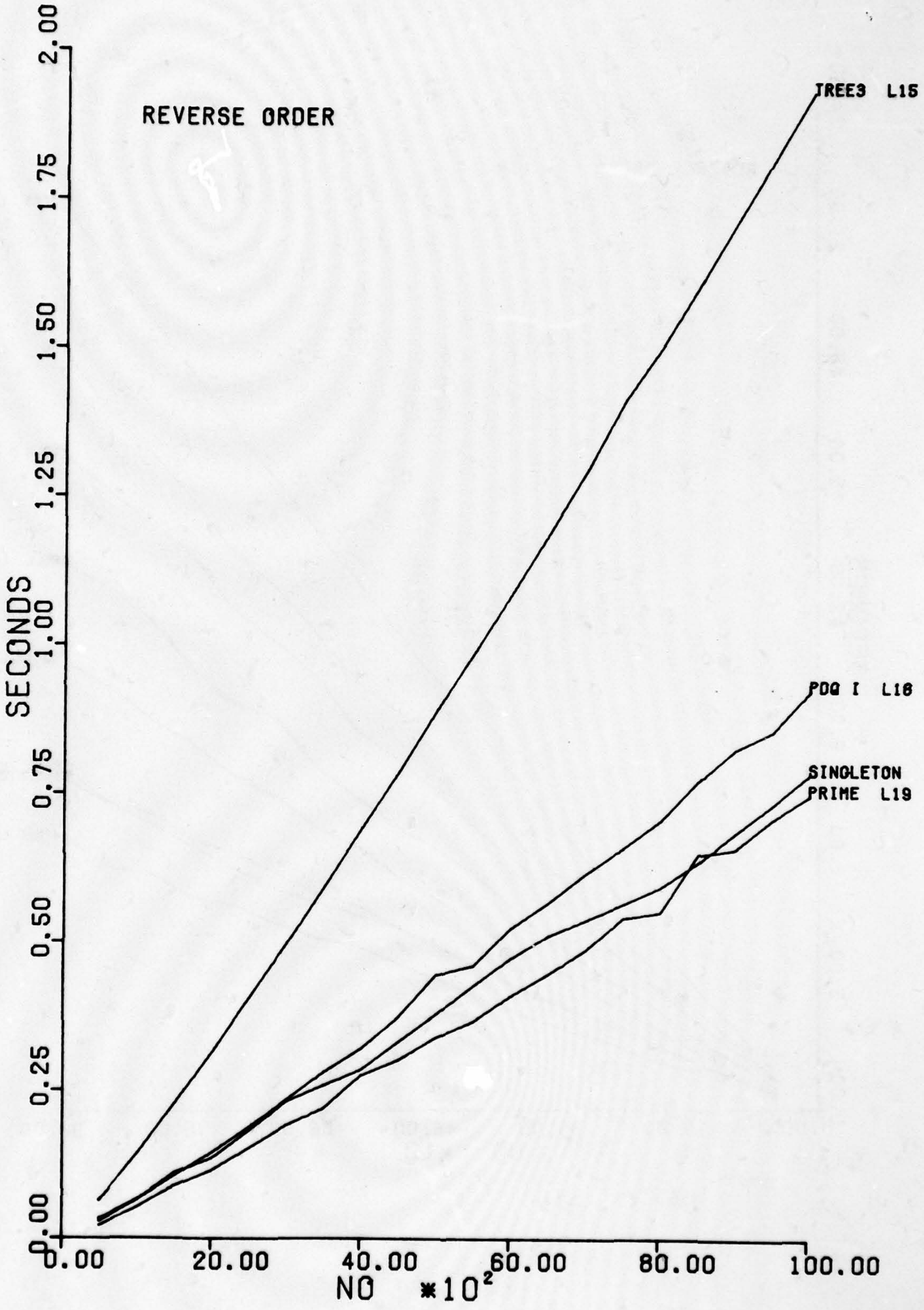

SORTED ARRAYS
TREES L15
PDQ I L16
SINGLETON
PRIME L19
SECONDS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00
NO $*10^2$

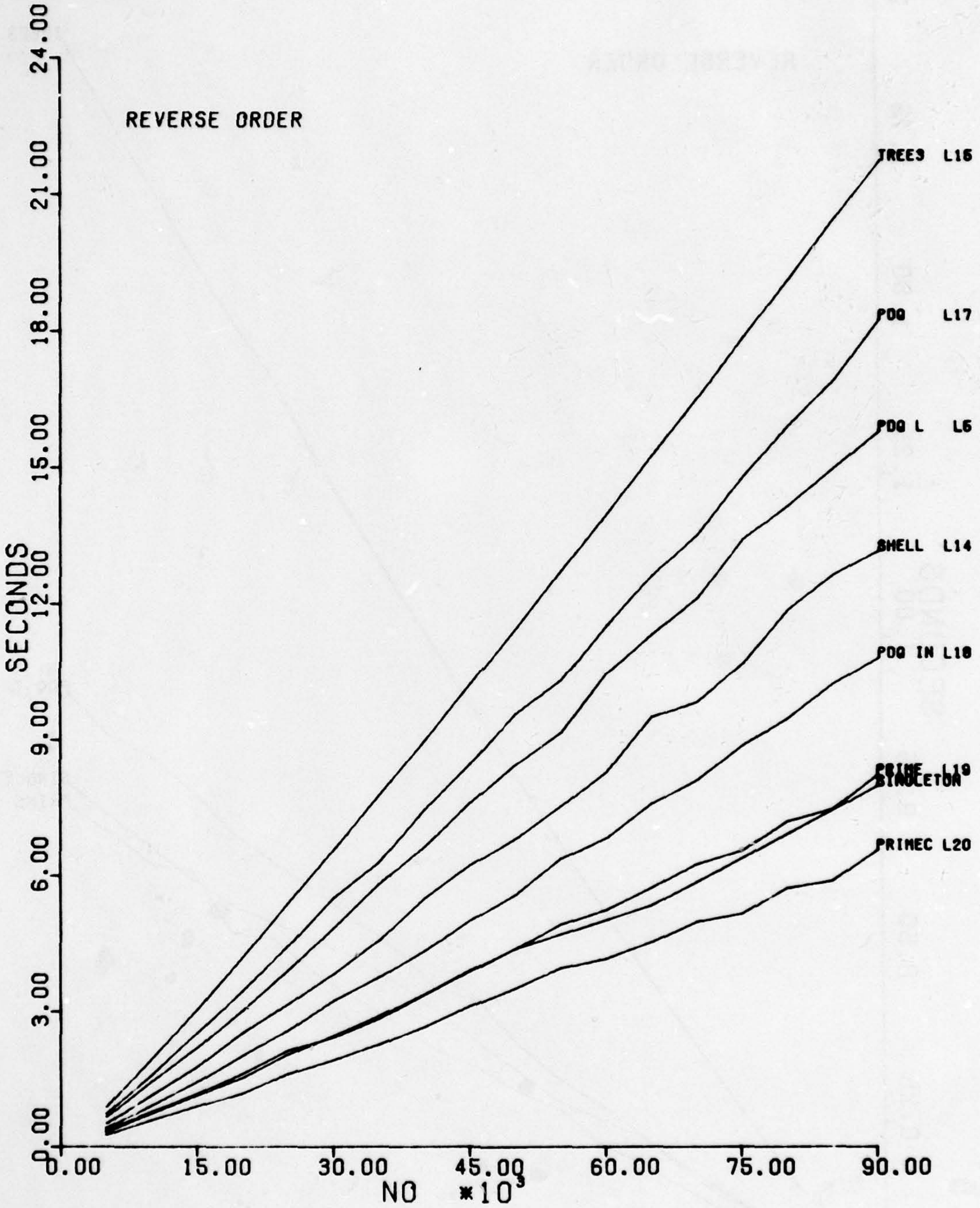

SORTED ARRAYS
SECONDS
NO *10³
0.00
3.00
6.00
9.00
12.00
15.00
18.00
21.00
24.00
15.00
30.00
45.00
60.00
75.00
90.00
TREES L16
PDS L17
PDS L L5
PDS IN L18
SHELL L14
SINGLETON
PRIME L19
PRIMEC L20

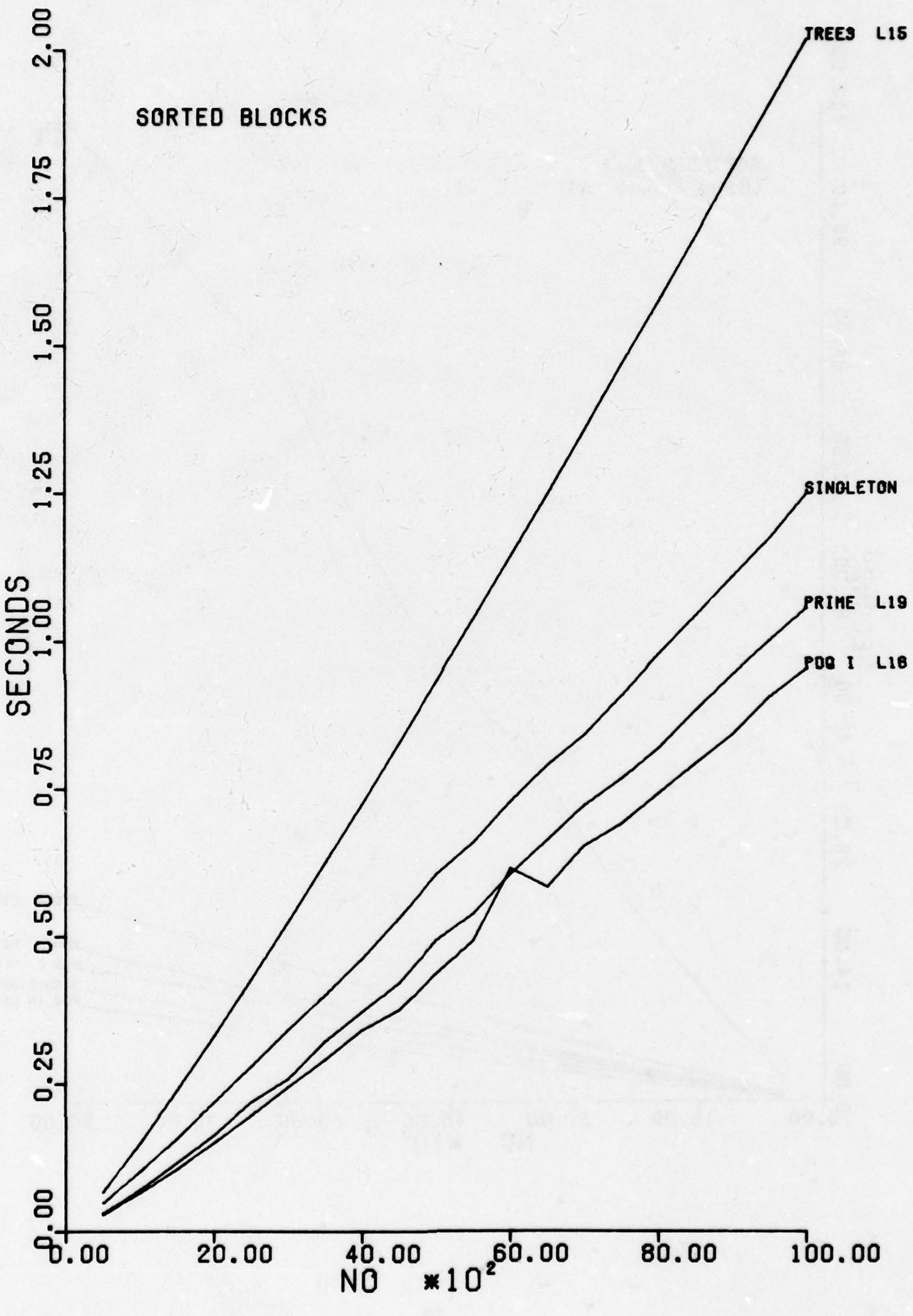

ALMOST SORTED
TREES L16
PRIME L19
SECONDS
NO *10²
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00



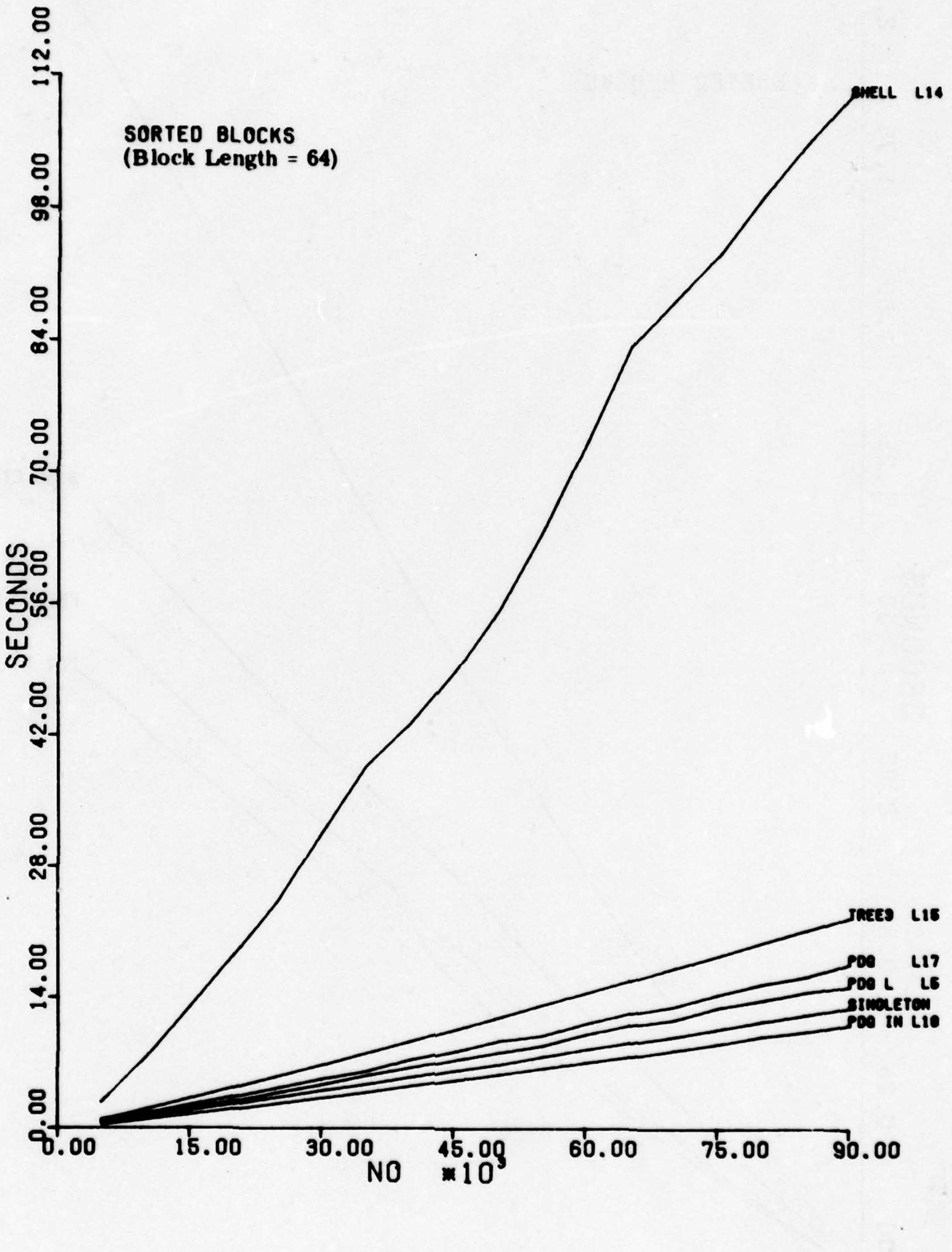
ALMOST SORTED
SECONDS
0.00
3.00
6.00
9.00
12.00
15.00
18.00
21.00
24.00
0.00
15.00
30.00
45.00
60.00
75.00
90.00
NO *10³
TREES L15
PDG L17
PDG L L5
SHELL L14
SINGLETON
PRIME L19
PRIMEC L20

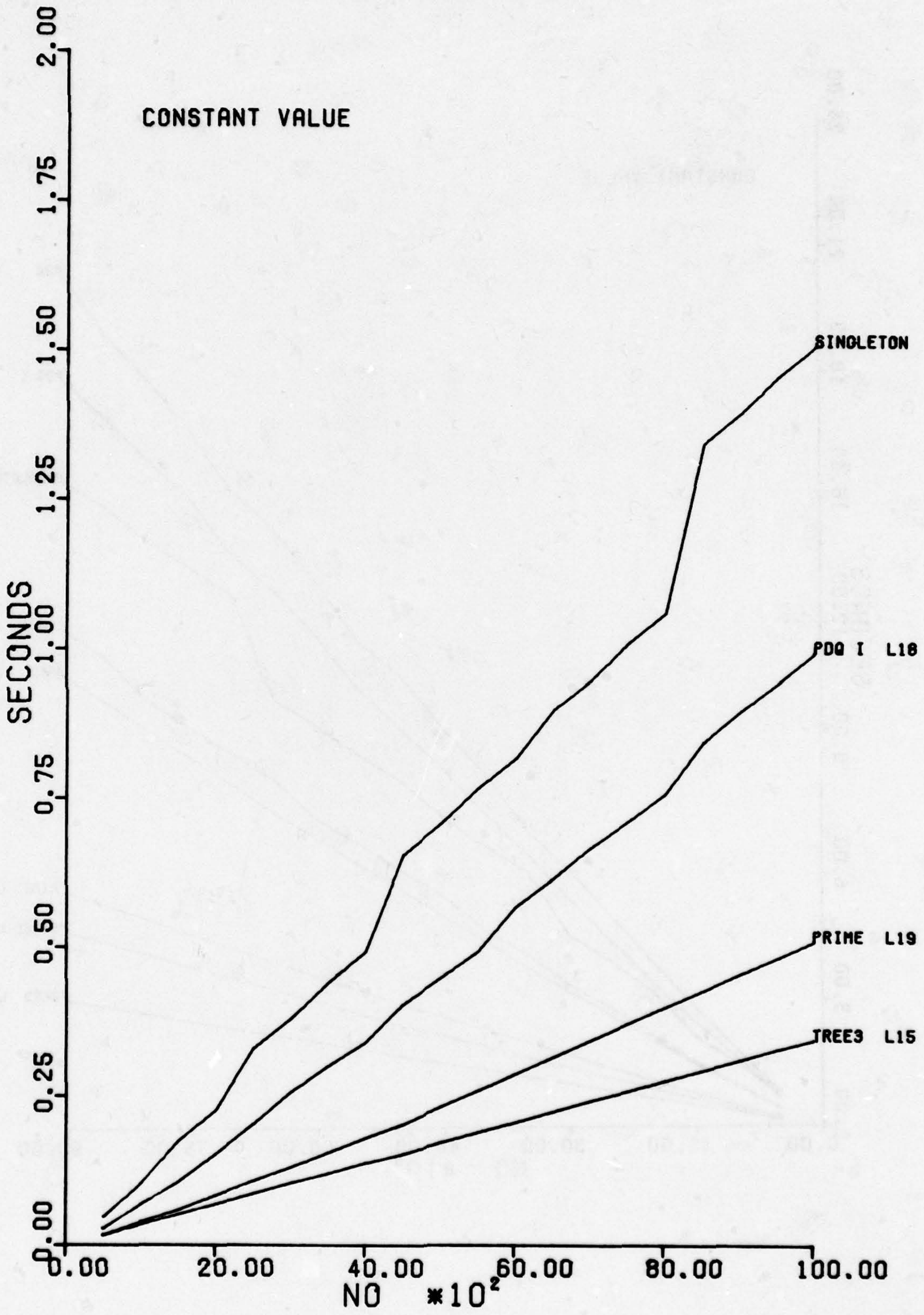

REVERSE ORDER
TREE3 L15
POG I L18
SINGLETON
PRIME L19
SECONDS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00
NO *10²

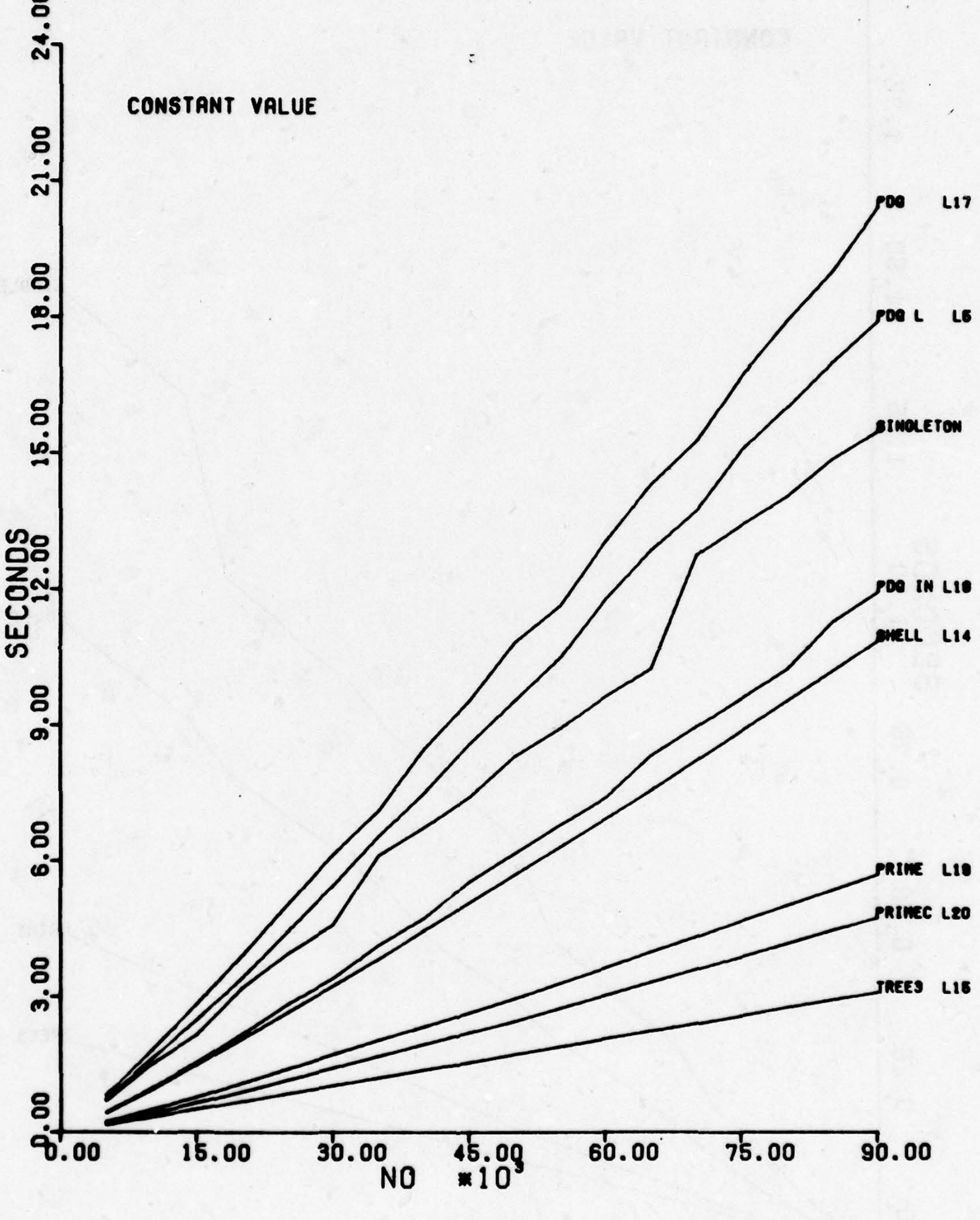

REVERSE ORDER
SECONDS
0.00
3.00
6.00
9.00
12.00
15.00
18.00
21.00
24.00
0.00
15.00
30.00
45.00
60.00
75.00
90.00
NO *10³
TREES L16
PDQ L17
PDQ L L6
SHELL L14
PDQ IN L18
PRIME L19
SINGLETON
PRIMEC L20



SORTED BLOCKS
TREES L15
SINGLETON
PRIME L19
PDQ I L18
SECONDS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00
NO *10²


SORTED BLOCKS
(Block Length = 64)
SECONDS
NO ＊10³
0.00
14.00
28.00
42.00
56.00
70.00
84.00
98.00
112.00
15.00
30.00
45.00
60.00
75.00
90.00
SHELL L14
TREES L16
PDG L17
PDG L L5
SINGLETON
PDG IN L18


CONSTANT VALUE
SECONDS
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.00
20.00
40.00
60.00
80.00
100.00
NO $*10^2$
SINGLETON
PDQ I L18
PRIME L19
TREE3 L15


CONSTANT VALUE
SECONDS
NO *10³
0.00
3.00
6.00
9.00
12.00
15.00
18.00
21.00
24.00
0.00
15.00
30.00
45.00
60.00
75.00
90.00
PDQ L17
PDQ L L6
SINGLETON
PDQ IN L18
SHELL L14
PRIME L19
PRIMEC L20
TREES L16

DISTRIBUTION LIST

Aeronautical Systems Division (AFSC)
Wright-Patterson AFB, OH 45433
Attn: ASD/ENESH (P. T. Marth)
Attn: ASD/ENFTV (D. J. Wallick) (2 copies)
Attn: ASD/XROL (F. Campanile)
Attn: ASD/XROL (R. K. Frick)
Attn: ASD/XROT (G. B. Bennett)
Attn: ASD/XRE (S. E. Tate)
Attn: ASD/YPES (C. Gebhard)

Aerospace Medical Research Laboratories
AMRL/MEA
Area B, Bldg. 33
Wright-Patterson AFB, OH 45433
Attn: AMRL/MEA (CAPT G. J. Valentino)

Aerospace Rescue and Recovery Service (MAC)
Scott AFB, IL 62225
Attn: ARRS/DOQ (CAPT J. J. Draham, Jr.)

Air Force Acquisition Logistics Division
Wright-Patterson AFB, OH 45433
Attn: AFALD/PTEA (MAJ D. Waltman)

Air Force Avionics Laboratory
Wright-Patterson AFB, OH 45433
Attn: AFAL/WRA-1 (E. Leaphart)
Attn: AFAL/WRP (W. F. Bahret)

Air Force Flight Dynamics Laboratory
Wright-Patterson AFB, OH 45433
Attn: AFFDL/FES (CDIC) (2 copies)
Attn: AFFDL/FES (C. W. Harris)
Attn: AFFDL/FES (J. Hodges)
Attn: AFFDL/FES (R. W. Lauzze)
Attn: AFFDL/FES (D. W. Voyls)
Attn: AFFDL/FGL
Attn: AFFDL/TST (Library)

Air Force Logistic Command
Wright-Patterson AFB, OH 45433
Attn: AFLC/LOE (Commander)

Air Force Systems Command
Andrews AFB, DC 20334
Attn: AFSC/DLCAA (P. L. Sandler)

Air Force Test and Evaluation Center
Kirtland AFB, NM 87115
Attn: AFTEC/OAR (K. Campbell)
Attn: AFTEC/OAR (R. Blankert)

Air Force Weapons Laboratory
Kirtland AFB, NM 87117
Attn: AFWL/PGV (CAPT J. K. Carson)
Attn: AFTEC/OA (MAJ H. Rede)

Applied Technology Laboratory
Army Research & Technology Laboratory (AVRADCOM)
Ft. Eustis, VA 23604
Attn: DAVDL-ATL-AL (Mr. Merritt)
Attn: DAVDL-ATL-ASV (S. Pociluyko)
Attn: DAVDL-ATL-ASV (H. W. Holland)
Attn: DAVDL-ATL-ASV (J. T. Robinson)

Armament Development and Test Center
Eglin AFB, FL 32542
Attn: ADTC/DLODL (Technical Library)
Attn: ADTC/XR (C. T. Maney)
Attn: ADTC/XRSP (E. Blair)

Armament Research and Development Command
Dover Base
Dover, NJ 07801
Attn: DRDAR-LCS (S. K. Einbinder)

Army Aviation Research & Development Command
P.O. Box 209
St. Louis, MO 63166
Attn: DRCPM-ASE-TM (MAJ Schwend) (2 copies)

Army Ballistic Research Laboratory
Aberdeen Proving Ground, MD 21005
Attn: DRDAR-BLV (D. W. Mowrer)

Army Foreign Science and Technology Center
220 Seventh St., NE
Charlottesville, VA 22901
Attn: DRXST-BA3 (E. R. McInturff)
Attn: DRXST-CA2 (J. M Blake) (2 copies)

Army Materials and Mechanics Research Center
Watertown, MA 02172
Attn: DRXMR-ER (F. C. Quigley)
Attn: DRXMR-EM (C. F. Hickey, Jr.)
Attn: DRXMR-PL (M. M. Murphy) (2 copies)
Attn: DRXMR-XC (E. S. Wright)

Army Materiel Systems Analysis Activity
Aberdeen Proving Ground, MD 21005
Attn: DRXSY-AA (Director)
Attn: DRXSY-AD (H. X. Peaker)
Attn: DRXSY-J (J. J. McCarthy)
Attn: DRXSY-S (J. R. Lindenmuth)

Chief of Naval Operations, Office of the
Room 5C735, Pentagon
Washington, DC 20350
Attn: OPNAV-982E3L (LT COL W. A. Allanson)

David W. Taylor Naval Ship R&D Center
Carderock Laboratory
Bethesda, MD 20084
Attn: Code 1740.2 (F. J. Fisch)
Attn: Code 1740.2 (F. Hackett)
Attn: Code 1740.4 (M. L. Salive)
Attn: Code 522 (Commander) (4 copies)

Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209
Attn: S. Zakanycz DARPA/STO

Defense Documentation Center
Cameron Station, Bldg. 5
Alexandria, VA 22314
Attn: DDC-TCA (2 copies)

Defense Systems Management College
Ft. Belvoir, VA 22060
Attn: W. Schmidt

Department of Transportation - FAA
2100 Second St., SW, Rm 1400C
Washington, DC 20591
Attn: ARD-520 (R. A. Kirsch)

Deputy Chief of Staff (AIR)
Marine Corps Headquarters
Washington, DC 20380
Attn: APW-22 (MAJ K. Van Esselstyn)

ERADCOM
Fort Monmouth, NJ 07703
Attn: DAVAA-I (S. Zywotow, Avionics R&D Activity)
Attn: DELEW-P (R. F. Giordano, Electronic Warfare Laboratory)
Attn: DELSD-E (C. Goldy)
Attn: DRSEL-GG-TD (Commander)

Foreign Technology Division (AFSC)
Wright-Patterson AFB, OH 45433
Attn: FTD/NICD (2 copies)

HQ Air Logistics Command
McClellan AFB, CA 95652
Attn: SM/MMSRBC (D. E. Snider)

Joint Strategic Target Planning Staff
Offutt AFB, NB 68113
Attn: JSTPS/JPTB (MAJ C. O. Cox)
Attn: NRI/STINFO (Library)

NASA - Ames Research Center
Army Research and Technology Laboratories, Headquarters
Mail Stop 207-5
Moffett Field, CA 94035
Attn: DAVDL-AS (V. L. J. Di Rito)

NASA - Johnson Space Center
Houston, TX 77058
Attn: EC (F. S. Dawn)

Naval Air Development Center
Warminster, PA 18974
Attn: Code 097 (MAJ W. Boeck)
Attn: Code 701 (B. Vafkos)
Attn: Code 2012 (C. E. Murrow)
Attn: Code 2012 (M. C. Mitchell)
Attn: Code 2012 (R. H. Beliveau)
Attn: Code 3023 (L. M. Rakszawski)
Attn: Code 6083 (S. L. Huang)
Attn: Code 601A (R. A. Ritter)

Naval Air Propulsion Center
P.O. Box 7176
Trenton, NJ 08628
Attn: PE3 (D. Wysocki)

Naval Air Systems
Airtevron One
Patuxent River, MD 20653
Attn: LT R. N. Freedman

Naval Air Systems Command
Washington, DC 20361
Attn: AIR-03PA4 (T. S. Momiyama)
Attn: AIR-330B (E. A. Lichtman)
Attn: AIR-350 (E. M. Fisher)
Attn: AIR-503W1 (E. A. Thibault)
Attn: AIR-52014 (L. Sztan)
Attn: AIR-5184 (D. Atkinson) (2 copies)
Attn: AIR-5184J (MAJ R. A. Horton)
Attn: AIR-5303
Attn: AIR-530313 (R. D. Hume)
Attn: AIR-53051A (P. Kicos)
Attn: AIR-5323A
Attn: AIR-5323K (K. Chang)
Attn: AIR-5323Z (S. Englander)
Attn: AIR-620B1 (LCDR K. K. Miles)
Attn: AIR-954 (Tech. Library)
Attn: PMA-2692A1 (R. W. Wills)

Naval Postgraduate School
Monterey, CA 93940
Attn: Code 67BP (R. E. Ball)
Attn: Library

Naval Research Laboratory
4555 Overlook Ave., SW
Washington, DC 20375
Attn: Code 1409 (J. M. MacCallum)
Attn: Code 5730 (E. E. Koos)

Naval Sea Systems Command
Washington, DC 20362
Attn: SEA-6543 (F. W. Sieve)

Naval Surface Weapons Center
Dahlgren Laboratory
Dahlgren, VA 22448
Attn: CK-2301 (J. E. Mitchell)
Attn: CN-61 (J. S. Nerrie)
Attn: G-10 (J. E. Ball)
Attn: G-10 (S. Hock)
Attn: G-10 (F. J. Petranka)
Attn: Library

Naval Surface Weapons Center
White Oak Laboratory
Silver Spring, MD 20910
Attn: CN-13
Attn: WX-21 (Library)

Naval War College
Newport, RI 02840
Attn: Code E-111 (President)

Naval Weapons Center
China Lake, CA 93555
Attn: Code 317 (M. H. Keith)
Attn: Code 3181 (C. Padgett) (2 copies)
Attn: Code 3183 (C. Driussi)

Naval Weapons Engineering Support Activity
Systems Analysis Dept.
Bldg. 210-2 (ESA-19)
Washington Navy Yard
Washington, DC 20374
Attn: Code ESA-1923 (C. W. Stokes III) (2 copies)
Attn: Code 11621 (J. Stasko)

Naval Weapons Support Center
Crane, IN 47522
Attn: Code 502 (N. L. Papke)

Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217
Attn: ONR 474 (N. Perrone)

Pacific Missile Test Center
Point Mugu, CA 93042
Attn: Code 4253-3 (Naval Air Station, Technical Library) (2 copies)

Air Force Logistics Center
Robins AFB, GA 31098
Attn: WRALC/MMETE (LT W. Shelton)

Aerojet ElectroSystems Company
A Div. of Aerojet-General Corp.
1100 W. Hollyvale Street
Azusa, CA 91702
Attn: A. R. Moorman

AiResearch Manufacturing Co. of Arizona
A Division of the Garrett Corp.
P.O. Box 5217
Phoenix, AZ 85010
Attn: G. L. Merrill

Armament Systems, Inc.
712-F North Valley Street
Anaheim, CA 92801
Attn: J. Musch

A. T. Kearney and Company, Inc.
100 South Wacker Drive
Chicago, IL 60606
Attn: R. H. Rose

AVCO
Lycoming Division
550 So. Main St.
Stratford, CT 06497
Attn: R. Cuny

Battelle Memorial Institute
505 King Ave.
Columbus, OH 43201
Attn: J. H. Brown, Jr.

The BDM Corp.
2600 Yale Blvd SE.
Albuquerque, NM 87106
Attn: A. J. Holten

The BDM Corp.
1920 Aline Ave.
Vienna, VA 22180
Attn: J. W. Milenski

Beech Aircraft Corp.
9709 E. Central Ave.
Wichita, KS 67201
Attn: Engineering Library

Bell Helicopter Textron
Division of Textron Inc.
P.O. Box 482
Fort Worth, TX 76101
Attn: Security/Dept. 12, J. R. Johnson

The Boeing Aerospace Company
P.O. Box 3999
Seattle, WA 98124
Attn: J. G. Avery, M/S 41-37
Attn: R. J. Helzer, M/S 47-28

The Boeing Company
Vertol Division
Boeing Center
P.O. Box 16848
Philadelphia, PA 19142
Attn: J. E. Gonsalves, M/S P32-19 (2 copies)

Boeing Wichita Company
3801 S. Oliver St.
Wichita, KS 67210
Attn: H. E. Corner, M/S K16-67
Attn: D. Y. Sink, M/S K16-14

Booz.Allen Applied Research
4330 East West Highway
Bethesda, MD 20014
Attn: W. Djinis

Calspan Corp.
P.O. Box 235
Buffalo, NY 14221
Attn: Library (V. M. Young)

COMARCO inc
1417 N. Norma
Ridgecrest, CA 93555
Attn: G. Russell (2 copies)

Denver Research Institute
University of Denver
University Park Station
2360 S. Gaylord Street
Denver, CO 80210
Attn: C. R. Hoggatt
Attn: R. F. Recht

ETI Effects Technology, Inc.
A Subsidary of Flow General, Inc.
5383 Hollister Avenue
Santa Barbara, CA 93111
Attn: Library Acquisitions (S. Clow)

Fairchild Industries, Inc.
Fairchild Republic Division
Conklin Ave.
Farmingdale, L.I., NY 11735
Attn: J. A. Arrighi
Attn: G. Mott
Attn: D. C. Watson
Attn: Engineering Library (G. A. Mauter)

Falcon Research and Development Co.
2350 Alamo Ave., SE
Albuquerque, NM 87106
Attn: W. L. Baker

Falcon Research and Development Co.
696 Fairmount Ave.
Baltimore, MD 21204
Attn: J. A. Silva

Firestone Tire & Rubber Co.
Firestone Coated Fabric Co. Division
P.O. Box 869
Magnolia, AR 71753
Attn: D. L. Byerley

Ford Aerospace and Communications Corp.
Ford Road, P.O. Box A
Newport Beach, CA 92663
Attn: Library

General Dynamics Corp.
Convair Division
P.O. Box 80986
San Diego, CA 92138
Attn: Research Library, MZ 40-6540 (U. J. Sweeney)

General Dynamics Corp.
Fort Worth Division
Grants Lane, P.O. Box 748
Fort Worth, TX 76101
Attn: P. R. deTonnancour/G. W. Bowen

General Electric Co.
Aircraft Engine Group
1000 Western Ave.
West Lynn, MA 01910
Attn: J. M. Wannemacher

General Electric Co.
Aircraft Engine Business Group
Evendale Plant
Mail Drop H-9
Cincinnati, OH 45215
Attn: AEG Technical Information Center (J. J. Brady)
Attn: B. Alexander (2 copies)

General Research Corporation
P.O. Box 6770
5383 Hollister Ave.
Santa Barbara, CA 93111
Attn: J. H. Cunningham
Attn: R. Rodman

General Research Corporation
SWL Division, Suite 700, Park Place
7926 Jones Branch Dr.
McLean, VA 22101
Attn: T. King

Goodyear Aerospace Corp.
Engineered Fabrics Division
1210 Massillon Rd.
Akron, OH 44315
Attn: Library, D/152G (R. L. Vittitoe/J. R. Wolfersberger) (3 copies)

Grumman Aerospace Corp.
South Oyster Bay Rd.
Bethpage, NY 11714
Attn: J. P. Archey Jr., Dept. 662, Mail C42-05
Attn: R. W. Harvey, Mail C27-05
Attn: H. L. Henze, B27-07
Attn: J. Noack, M/S B10-25
Attn: Technical Information Center, Plant 35 L01-35 (H. B. Smith)

Hughes Helicopters
A Division of Summa Corp.
Centinela Ave. & Teal St.
Culver City, CA 90230
Attn: Library, 2/T2124 (D. K. Goss)

Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202
Attn: Technical Information Center, DIMO (P. Okamoto)

IIT Research Institute
10 West 35 Street
Chicago, IL 60616
Attn: I. Pincus

The Johns Hopkins University
Applied Physics Laboratory
Johns Hopkins Road
Laurel, MD 20810
Attn: C. F. Meyer
Attn: V. W. Woodford

Kaman Aerospace Corporation
Old Windsor Rd.
Bloomfield, CT 06002
Attn: H.E. Showalter

Lockheed-California Co.
A Division of Lockheed Aircraft Corp.
2555 Hollywood Way
P.O. Box 551
Burbank, CA 91520
Attn: Technological Information Center, 84-40 Unit 35, Plant A-1
Attn: G. E. Raymer, D/75-84 Bldg. 63 A-1 (2 copies)
Attn: A. D. Jackmond, Dept. 75-60, Bldg. 170 B-1

Lockheed-Georgia Co.
A Division of Lockheed Aircraft Corp.
86 S. Cobb Drive
Marietta, GA 30063
Attn: W. T. Mikolowsky, 72-08 Zone 415
Attn: Sci-Tech Info Center, 72-34 Zone 26 (T. J. Kopkin)

Lockheed Missiles & Space Company, Inc.
A Subsidiary of Lockheed Aircraft Corp.
P.O. Box 504
Sunnyvale, CA 94088
Attn: G. R. Evans (5501-572-5)

Martin Marietta Corp.
Orlando Division
P.O. Box 5837
Orlando, FL 32855
Attn: Library (M. C. Griffith, MP-30)

McDonnell Douglas Corp.
Douglas Aircraft Company
3855 Lakewood Blvd.
Long Beach, CA 90846
Attn: Technical Library, C1-250/36-84 AUTO 14-78 (3 copies)

McDonnell Douglas Corp.
P.O. Box 516
St. Louis, MO 63166
Attn: R. D. Detrich, Dept. 022

New Mexico Institute of Mining and Technology
Campus Station
Socorro, NM 87801
Attn: Tera

Northrop Corp.
Aircraft Division
3901 W. Broadway
Hawthorne, CA 90250
Attn: J. H. Bach, 2130/83
Attn: V. B. Bertagna, 3451/81
Attn: H. W. Jones, 3360/82
Attn: J. F. Paris, 3628/83

Northrop Corp.
Ventura Division
1515 Rancho Conejo Blvd.
P.O. Box 2500
Newbury Park, CA 91320
Attn: M. Raine

The Rand Corp.
1700 Main St.
Santa Monica, CA 90406
Attn: N. W. Crawford

R&D Associates
P.O. Box 9695
Marina Del Rey, CA 90291
Attn: B. Jaeger

Rockwell International Corp.
Los Angeles Division
International Airport
5701 W. Imperial Hwy
Los Angeles, CA 90009
Attn: R. Hurst, MB56
Attn: W. L. Jackson
Attn: S. C. Mellin, AD-25
Attn: R. Moonan, AB78 (2 copies)
Attn: T. C. Getten, MB56

Rockwell International Corp.
4300 E. Fifth Ave.
P.O. Box 1259
Columbus, OH 43216
Attn: Technical Information Center (D. Z. Cox) (2 copies)

Science Applications, Inc.
200 Lomas Blvd., N. W., Suite 1020
Albuquerque, NM 87102
Attn: Library

Science Applications, Inc.
2361 Jefferson Davis Hwy.
Arlington, VA 22202
Attn: D. A. Venor

Sikorsky Aircraft Division
United Technologies Corp.
North Main Street
Stratford, CT 06002
Attn: D. P. Bartz, Chief Survivability

Southwest Research Institute
P.O. Drawer 28510
San Antonio, TX 78284
Attn: Bessey, Div. 02
Attn: P. H. Zabel, Div. 02

Stanford Research Institute
333 Ravenswood Ave.
Menlo Park, CA 94025
Attn: J. Golins

System Planning Corporation
1500 Wilson Blvd., Suite 1300
Arlington, VA 22209
Attn: J. A. Navarro

Telcom Systems, Inc.
A subsidiary of Telcom Incorporated
2300 South 9th St.
Suite 400
Arlington, VA 22204
Attn: C. Gentzel

Teledyne Ryan Aeronautical
2701 Harbor Dr.
San Diego, CA 92112
Attn: Technical Information Services (W. E. Ebner)

Uniroyal Inc.
312 N. Hill Street
Mishawaka, IN 46544
Attn: J. D. Galloway

United Technologies Corporation
Government Products Division
P.O. Box 2691
West Palm Beach, FL 33402
Attn: P. E. Desrosiers, Security Officer
Attn: J. Fyfe, Mail E-39

University of Dayton
Research Institute
300 College Ave.
Dayton, OH 45409
Attn: M. Goldschmidt

Vought Corporation
P.O. Box 225907
Dallas, TX 75265
Attn: D. M. Reedy, 2-30100

Williams Research Corp.
2280 W. Maple Rd.
Walled Lake, MI 48088
Attn: Library

ABSTRACT CARD

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

ABSTRACT CARD

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

Aeronautical Systems Division

PRIME and PDQ Sorts – Efficient Minimal Storage Sorting Algorithms, by R. R. Hilbrand, Wright-Patterson AFB, OH, ASD, for Joint Technical Coordinating Group/Aircraft Survivability, August 1979. 54 pp. (JTCG/AS-78-V-004, publication UNCLASSIFIED.)

One of the problems involved in computer programs for vulnerability assessment is that of rapidly sorting and arranging large sets of data. Two sorting algorithms, designated PRIME and PDQ, have been developed at ASD to more efficiently perform this function in vulnerability programs such

(Over)
1 card, 8 copies

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.

JTCG/AS-78-V-004

as SESTEM and FASTGEN II. The results are compared to those obtained with three other algorithms, SHELLSORT, TREESORT3, and SINGLETON. The newly developed sorts are shown to be significantly faster on the ASD CDC 6600 computer than the existing sorts. When used in an ASD missile endgame model SESTEM, the average run time was reduced by 20 to 25%. Program listings, flow charts, and typical output data are presented.